

بسم الله الرحمن الرحيم

كتاب نظم التشغيل الإلكتروني

بسم الله والحمد لله والصلاة والسلام على رسول الله ... يعتبر هذا الكتاب خلاصة جهد طالبات مادة نظم التشغيل في قسم تقنية المعلومات بجامعة الملك سعود خلال الفصل الدراسي الأول من العام الدراسي 1429/1428 هـــ، حيث تولى فريق من المتطوعات جمعه وتنسيقه ومن ثم إخراجه على هيئة كتاب إلكتروني ليكون متاح للجميع.

قامت الطالبات عبر فصول الكتاب بشرح المفاهيم الأساسية في مادة نظم التشغيل بلغة عربية سلسة مدعمة برسوم توضيحية مبتكرة ومستندين بذلك على فهم المحاضرة والكتاب المقرر وبعض المصادر على الشبكة العنكبوتية.

الكتاب لازال في مرحلة الألفا (Alpha) بمعنى أنه بحاجة لمزيد من التنقيح والمراجعة والتدقيق، ولكن لا يمنع ذلك من الاستفادة من مكنوناته والتبحر في محتوياته.

يتناول الكتاب في فصوله الإحدى عشرة المواضيع التالية:

- مقدمة تمهيدية: تاريخ نظم التشغيل
- الفصل الأول: نظرة عامة على نظم التشغيل
- الفصل الثاني: هيكلة نظم التشغيل
- الفصل الثالث: العمليات
- الفصل الرابع: الخيوط
- الفصل الخامس: جدولة وحدة المعالجة المركزية
- الفصل السادس: الجمود
- الفصل السابع: إدارة الذاكرة والذاكرة التخيلية

- الفصل الثامن: نظام الملفات
- الفصل التاسع: تراكيب وحدات التخزين
- الفصل العاشر: أنظمة الإدخال والإخراج.

ختاماً مع تقديرنا لمثل هذا الجهد المتميز ننتظر من القراء مشاركتهم بآرائهم وانتقاداتهم للخروج بالكتاب بصورة أفضل، فهذا العمل ما هو إلا جهد مقل في سبيل نشر العلم وخدمة طلابه، نسأل الله أن ينفع به وأن يكون هذا العلم شاهداً لهم لا عليهم ويبارك لكل من كتب حرفاً في هذا الكتاب أو جمعه و نسقه.

والحمد لله أولاً وأخراً.

مقدمة تمهيدية:

تاريخ نظم التشغيل

تاريخ نظم التشغيل[1]

صاحب التطوّر الثوري لأجهزة الحاسب الآلي تطوراً مماثلاً لنظم التشغيل التي تعمل على هذه الأجهزة، لتعكس احتياجات المستخدم والأهداف التي ساندت تطور الجهاز أصلاً.

بدءاً من الأجهزة القديمة التي لم يكن نظام التشغيل أساساً لعملها، إذ يأتي المستخدم بالعمليات والبيانات مخزّنة على بطاقات أو أشرطة مغناطيسية، لتقوم الآلة بمعالجة هذا البرنامج إلى أن ينتهي أو يتم إيقافه.

وصولاً إلى أنظمة التشغيل الحديثة التي تمكّن المستخدم من تنفيذ الكثير من العمليّات ومن تحقيق العديد من الأهداف في وقت قياسي.

الأربعينات: برمجة بلغة الآلة

في بداية الأربعينات (1940م) كانت الأجهزة بدائية جداً وبسيطة بحيث لا تتطلب أنظمة تشغيل لعملها ذلك أنها تنهي برنامج واحد في وقت بطيء نسبيّاً قبل تمكنها من بداية أي برامج أخرى.

في البداية كان على المستخدم كتابة جميع العمليات بلغة الآلة، بحيث كان المستخدم هو المبرمج ، بالإضافة إلى كتابة عمليات الإدخال والإخراج بالتفصيل، لكن لم يمض زمن طويل قبل أن جُمعت جميع عمليات الإدخال والإخراج التي كانت في السابق تشكل صعوبة في بناء البرامج، جمعت هذه العمليات لتكوين مكتبة عمليات إدخال وإخراج سميت بـ (IOCS: Input/Output Control System) فعندما يرغب المبرمج إجراء عملية إدخال أو إخراج ما عليه إلا إجراء نداء لأحد الدوالّ التي تحتويها المكتبة لتقوم بالغرض. هذه الخطوة سهّلت الكثير من العقبات والصعوبات التي كان يمر بها مبرمجو تلك الفترة.

[1]إعداد: أفنان السبيهين

هذه المكتبة كانت النواة التي منها تطور مبدأ نظام التشغيل، إذ كانت المكتبة مخزّنة في الآلة لتسهيل عملها والعمل عليها.

في هذه الأنظمة قضى المستخدم وقتاً كبيراً في تحويل الجهاز من عملية إلى أخرى. إذ أنه عند انتهاء إحدى البرامج كان عليه إزالة جميع الأشرطة المغناطيسية المحتوية على بيانات البرنامج، وإزالة البطاقات التي تحتوي على الأوامر، ليضع بدلاً عنها الأشرطة التي تحتوي على البرنامج أو العملية القادمة، كحلّ لهذه المشكلة صممت معامل جنرال موتورز (General Motors Research Laboratories) لأجهزة الــ IBM 701 Mainframe في عام 1956م نظام ميكانيكي للتحويل من برنامج لآخر. لاقت هذه الطريقة نجاحاً كبيراً وكانت بداية أنظمة الباتش (Batch Computing).

الستينات: أنظمة الباتش والــMultiprogramming

ويقصد بطريقة الباتش في الحساب أن تجمع جميع الوظائف في مجموعة واحدة من البطاقات التي يتعرف عليها الجهاز، ويفصل بين كل وظيفة وأخرى بطاقة تحكم. تتحكم الأجهزة بهذه البطاقات المحتوية على الوظائف عن طريق لغة تدعى بـــ (JCL: Job Control Language) . وهكذا عند انتهاء بطاقة وظيفة يقرأ الجهاز بطاقة التحكم التي تليها المحتوية على معلومات تخص الوظيفة القادمة.

نظام المعالجة (الباتش) قدّم تطوراً كبيراً في أداء الأجهزة في ذلك الحين (عام 1960م) ، كما وضّح ضرورة استخدام نظام تشغيل لإدارة مكونات الجهاز.

في عهد الستينات تطورت أنظمة معالجة الباتش فسمحت بمعالجة أكثر من وظيفة في نفس الوقت نتيجة لملاحظة مصممو نظم التشغيل أنه عند طلب وظيفة ما لأمر إدخال أو إخراج فإن المعالج يظل في وضع انتظار لحين تلبية الطلب، لذلك كان على وظيفة أخرى الاستفادة من الوقت الذي ينتظره المعالج، إذ تتعاقب الوظائف بين المعالج وبين الانتظار لأمر إدخال أو أخراج. وسميت هذه الطريقة التي توفر أكبر استخدام للمعالج ولمصادر الإدخال والإخراج بـــ (Multiprogramming). ولا ننسى ملاحظة أنه عندما يستطيع النظام معالجة أكثر من وظيفة في نفس الوقت فإنه يمكنه خدمة أكثر من مستخدم في نفس الوقت !.المشكلة الوحيدة التي واجهتها الأنظمة في ذلك الوقت هو محدوديّة قدرة الأجهزة على تخزين الوظائف التي ستعمل عليها في نفس الوقت.

حينها تم تطوير أسلوب آخر يعزز من قابلية الأجهزة لخدمة أكثر من مستخدم في نفس الوقت، هذه الطريقة سميت بـــ (Timesharing) إذ يخصص المعالج لكل وظيفة وقت محدد للعمل عليها ثم ينتقل للوظيفة التالية وهكذا، شريحة الوقت المخصصة للعمل على كل وظيفة تم اختيارها بدقة حيث أنها مدة صغيرة جداً في إدراك البشر لكن يمكن للمعالج أن ينجز بها عملاً كثيراً، وهكذا يظن كل مستخدم بأن المعالج مكرّس لإنجاز برنامجه بينما هو في الواقع يعمل على أكثر من برنامج في نفس الوقت.

السبعينات: شبكات الحاسب والحاجة إلى الحماية

حتى الآن هذه البرامج ومبادئ نظم التشغيل لم تكن إلا بحوث احتوتها معامل الجامعات والشركات الكبرى، وحتى السبعينات (1970م) حيث تطوّر مبدأ تسويق البرامج ونظم التشغيل. ومما ساعد على تسويق هذه الأنظمة للشركات والجامعات ومختلف المنظمات الحكومية قابلية التواصل بين الأجهزة ونقل البيانات (TCP/IP) .

ونتيجة للاستخدام الواسع (على مستوى الحكومة والجامعات وليس على مستوى العامّة) للتواصل في البيانات واستخدام الشبكات احتاجت نظم التشغيل في هذا العقد إلى تطوير إمكانياتها الشبكية و كذلك إلى تطوير نظم الأمن والحماية فيها، فكان الهدف في تلك الفترة إيجاد نظام تشغيل آمن ومحمي ضد الفيروسات والمتدخّلين.

شهدت تلك الفترة تطور نظام التشغيل اليونكس (UNIX) ، في بادئ الأمر كان نظام اليونكس مثل أي نظام آخر من حيث أنه كان يعتمد على الجهاز إذ تم كتابته بلغة التجميع Assembly Language)) محتوياً العيب الذي تحتويه جميع نظم التشغيل في تلك الفترة، لكن تم تصميم لغة الـــ C خصيصاً لحل هذا العيب في نظام اليونكس. فأصبح نظام اليونكس أول نظام تشغيل تتم كتابته بلغة أعلى من لغة الآلة مما جعله نظاماً يعمل على جميع أنواع الأجهزة ويحتوي على الكثير من المميزات و الإمكانيات التي افتقرت لها الكثير من نظم التشغيل في تلك الفترة. وليس من المستغرب أن يجد هذا النظام إقبالاً شديداً من قبل الجامعات والمنظمات خصوصاً بعد دعم معامل بل له (Bell Laboratories).

كما شهدت فترة السبعينات التطوّر السريع في المعالجات التي تحتويها أجهزة الحاسب، تطوّر المعالجات صاحبه تطور في إمكانيات الحاسب بالتالي تطور في الخدمات التي يوفرها نظام التشغيل. كما كان تطور المعالجات النواة التي غذّت فكرة الحاسب الشخصي الذي انتشر انتشاراً واسعاً في التسعينات.

لكن نظام اليونكس لم يكن الاختيار الأفضل للمستخدم العادي وللأجهزة المترلية حين بدأت انتشارها في الثمانينات، إذ كان نظاماً ذو أوامر معقدة صعبة الفهم.

الثمانينات: ثورة الحاسب الشخصي

لعبت شركة آبل (أسسها كل من Steve Jobs و Steve Wozniakفي عام 1975م) دوراً كبيراً في انتشار فكرة الحاسب الشخصي إذ جعلت انتشاره هدفها الأساسي. حيث قدمت فكرة أن الحاسب عبارة عن صندوق ذو أسلاك كهربائية يمكن إدخالها في أي قابس يحتويه المترل العادي.

انتشار فكرة الحاسب الشخصي غذّت الثورة في نظم التشغيل. إذ لم يكن ينقص الحواسب إلا نظام تشغيل واضح وسهل الاستخدام لجذب المستخدم العادي. كما أن المعالجات المخصصة للحواسب الشخصية تتطلب نظم تشغيل خاصّة بها من حيث الإمكانيات التي يمكنها تقديمها للمستخدم.

في الثمانينات دخلت IBM عالم الحواسيب الشخصيّة مغيّرة بذلك مجرى تاريخ نظم التشغيل المكرسة لهذه الحواسيب. كانت أغلب الشركات مترددة من حيث دخولها لسوق الحواسب الشخصية حيث لم يظهر اهتمام الأشخاص والشركات العادية بهذه الأجهزة. لكن النجاح الذي لاقته شركة آبل (خاصة بعد دمجها برامج إدارة الأعمال مثل محرر البيانات والجداول مع نظام تشغيلها) أثر على نظرة بقيّة الشركات لهذا السوق وشجّع شركة IBM لاتخاذ قرارها. حيث أنتجت أجهزة مبنية حول أسرع معالج في تلك الفترة Intel's 16-bit 8080)) لكن كانت مشكلتها الوحيدة هو نظام التشغيل، مع أنها كانت أكبر شركة لإنتاج البرامج في تلك الفترة، لكنها خبرتها في مجال الأجهزة الشخصية كانت قليلة. حينها تعاقدت العقد الشهير مع Bill Gates لإنتاج نظام تشغيل خاص بأجهزتها. ولا يخفى علينا تأثير هذا العقد على مكانة Bill Gates بين أغنياء العالم إذ اشترط حصوله على مبلغ من 10 إلى 50 دولار عن كل نسخة تباع من نظام تشغيله!.

لكن وجد Bill Gates نفسه بلا نظام تشغيل ولا مصادر تمكنه من إنتاج واحد لشركة IBM حينها استعان بنظام تشغيل طوره Tim Paterson لمعالج 8080 وكان يدعى بـــ :QDOS (Quick and Dirty Operating System) أنفقت شركة Microsoft مبالغ طائلة للحصول على حقوق النظام وبعد إجراء تعديلات بسيطة تمت إعادة تسميته بـــ (MS-DOS) .
وبذلك في عام 1991م وبالتحديد في شهر أغسطس أصبح 1700 جهاز منزلي متوفراً للشراء بسعر في متناول الأشخاص العاديون.

المصدر:

http://www.computinghistorymuseum.org/teaching/papers/research/history_of_operating_system_Moumina.pdf

الفصل الثاني:

هيكلة نظم التشغيل

هيكلة نظم التشغيل

Operating System Structures

إن نظام التشغيل هو الأساس المتحكم في تصرفات أجهزة الحاسب الآلي , فهو يقوم بتوفير البيئة المناسبة لتنفيذ وتشغيل البرامج, أي أننا لن نستطيع العمل و تشغيل البرامج على أجهزتنا بدون تواجد نظام تشغيل في الجهاز يتحكم في العمليات المختلفة.

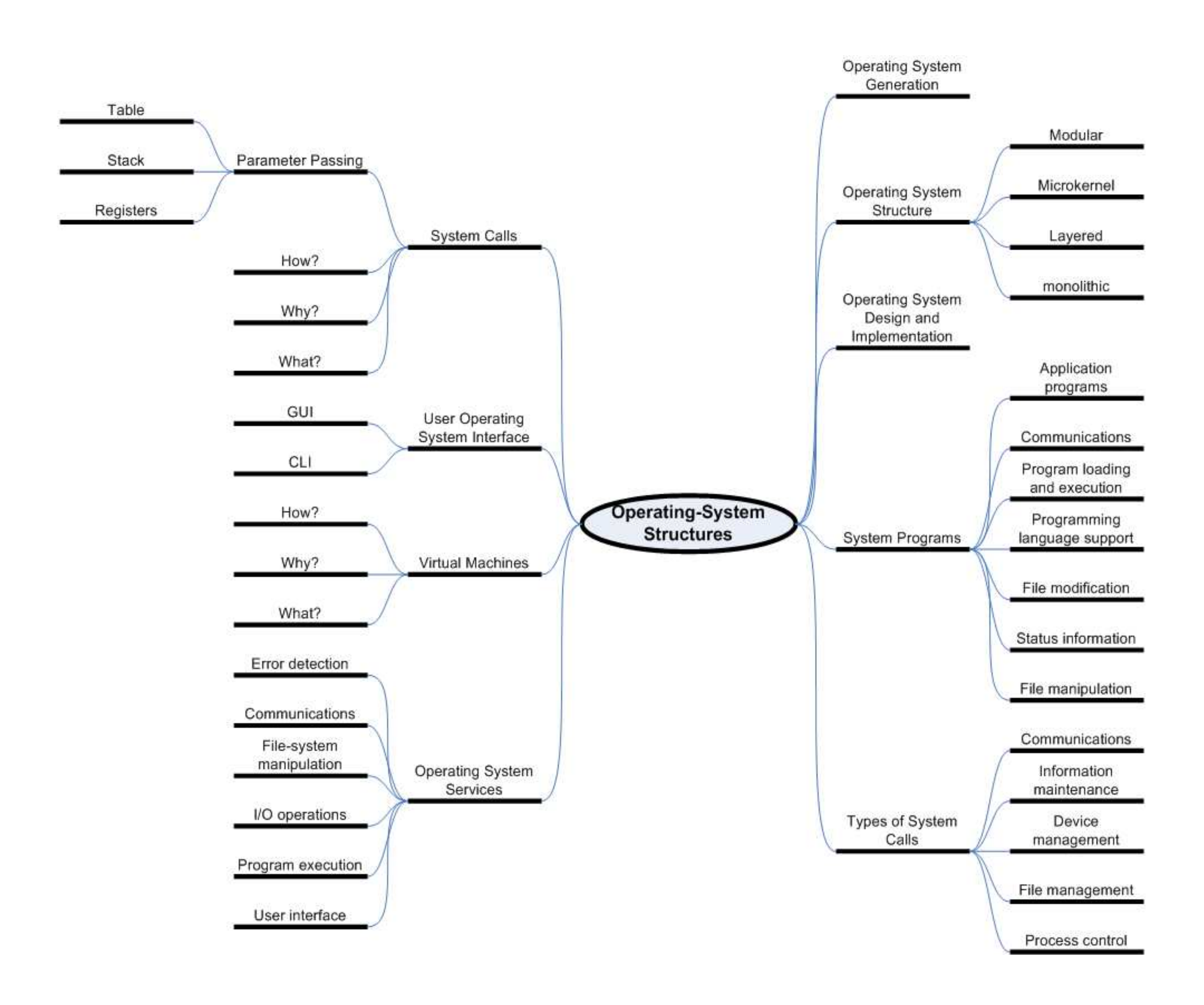
تُوفّر نظم التشغيل للمستخدمين العديد من الخدمات التي سنتطرق إليها في هذا الفصل. كما سنتحدث عن عدد من المفاهيم المهمة عن نظم التشغيل و تركيبها, وطرق تفاعلها مع المستخدمين.

أهداف هذا الفصل:

- التعرف على الخدمات التي توفرها نظم التشغيل المختلفة لمستخدميها.
- التعرف على الواجهات التي يتم من خلالها التفاعل بين المستخدم ونظام التشغيل.
- معرفة طرق الاتصال بين أجزاء نظام التشغيل.
- معرفة الطرق المختلفة لبناء نظام التشغيل.
- كذلك التعرف على بعض المفاهيم المهمة مثل الآلات الافتراضية و تنشئة النظام.

تم جمع هذا الفصل بواسطة قطرالندى عبدالرحمن السماعيل (xxdewdropsxx@gmail.com)

الخريطة الذهنية للفصل:



أولاً: خدمات نظام التشغيل[2](Operating System Services)

يوفر نظام التشغيل خدمات معينة للبرامج التي يقوم بتنفيذها كما يوفر خدمات لمستخدمي هذه البرامج، وبالطبع فإن توفير هذه الخدمات يختلف من نظام تشغيل إلى آخر ولكنها ترتبط في بعض الأوجه.

وسوف نتطرق في موضوعنا هذا لمجموعة من تلك الخدمات التي يوفرها نظام التشغيل.

أولاً: **خدمات نظام التشغيل التي تساعد المستخدم بشكل مباشر :**

1- واجهـــــة المستخـــدم(User Interface):

جميع أنظمة التشغيل تحتوي على واجهة للمستخدم وتأخذ هذه الواجهة أكثر من شكل, ومن أشكال واجهة المستخدم:

1. (Command Line Interface-CLI) أي الواجهة النصية.
2. (Graphical User Interface-GUI) أي الواجهة الرسومية، وهي الأكثر شيوعاً واستخداماً.

والجدير بالذكر أن بعض الأنظمة مزودة باثنين أو ثلاثة من الواجهات المختلفة.

2- تنفيذ البرامـــــج(Program Execution):

يجب أن يكون لنظام التشغيل قدرة كافية لتحميل البرامج في الذاكرة وتنفيذ تلك البرامج, ويجب أيضاً أن يكون مؤهلا لاختتام التطبيق بطريقة إما عادية أو غير عادية -عند وجود بعض الأخطاء-.

3- عمليات الإدخال والإخراج (I/O Operations):

إن أي برنامج يتم تطبيقه قد يكون بحاجة إلى عمليات إدخال وإخراج بحيث يقوم بطلب ملف معين أو أجهزة الإدخال والإخراج.

[2] إعداد: منار القحطاني, إيمان الزهراني, سمية الخنيزان

يجب أن يكون نظام التشغيل هو الوسيلة للقيام بالإدخال والإخراج وذلك لأن المستخدم لا يستطيع عادة أن يتحكم بالمدخلات والمخرجات مباشرة وذلك لحمايتها وزيادة الفعالية.

4- تشكيل نظـــام الملفـــات(File System Manipulation):

لنظام الملفات اهتمام خاص في نظم التشغيل. وذلك لأن البرامج تقوم بعمليات كثيرة على الملفات كقراءة وكتابة الملفات, وتكوين وحذف هذه الملفات والأدلة من خلال اسمها أو البحث عن ملف معين, وغيرها الكثير من العمليات التي تتم على الملفات والأدلة.

5- _الاتصالات (communications):

قد تحتاج العلميات في بعض الحالات للاتصال مع بعضها البعض لتبادل المعلومات والبيانات ومشاركتها فيما بينها. وهذا الاتصال قد يكون على نفس الحاسوب أو على حاسبات مختلفة عبر شبكة. وهذه المشاركة تتم بطريقتين هما : الذاكرة المشتركة (shared memory) أو عن طريق الرسائل العابرة (message passing) .

6- كشف الخطأ(Error Detection):

إن خطأ واحد من جزء من النظام قد يسبب عطل كامل في النظام! لتفادي مثل هذه المشكلة يقوم نظام التشغيل بمراقبة النظام بشكل مستمر لاكتشاف الأخطاء التي قد تحدث ويقوم بالإجراءات المناسبة لتصحيحها عند حدوثها.

ثانيا: مجموعة أخرى من خدمات نظام التشغيل موجودة لضمان كفاءة تشغيل النظام عبر تقاسم الموارد(resource sharing):

1- تخصيص الموارد (Resource allocation) :

إذا وُجد لدينا أكثر من مستخدم ، أو أكثر من عمل يتم تنفيذه بنفس الوقت ، يجب أن يتم تخصيص الموارد لكلٍ منهم ، وتوجد عدة أنواع من الموارد , بعضها تحتاج إلى تخصيص كود خاص مثل : الذاكرة الرئيسية وتخزين الملفات ، وأخرى كأجهزة الإدخال والإخراج تتطلب كود عام.

2– المحاسبة (Accounting) :

تستخدم هذه الخدمة من أجل تتبع المستخدمين ، ومعرفة أنواع الموارد المستخدمة من قبل كل مستخدم .

3– الحماية والأمن (Protection and security) :

الأشخاص الذين يمتلكون معلومات في أجهزة موصولة بشبكة أو في جهاز يستخدمه عدد من المستخدمين، يريدون ضمان حماية المعلومات ، وعدم تداخل العمليات التي تتم بنفس الوقت مع بعضها البعض .

المصادر:

- Operating system Concepts (Seventh Edition): Abraham Silberschatz,Peter Baer Galvin,Greg Gagne
- http://www.personal.kent.edu/~rmuhamma/OpSystems/Myos/sysService.htm

ثانياً: واجهة مستخدم نظام التشغيل[3](User Operating System Interface)

واجهة مستخدم نظام التشغيل هي الواجهة المرئية لمستخدمي النظام. وهي عبارة قشرة (**shell**) أو غلاف لنظام التشغيل. وهي برنامج يعمل في الطبقة العليا من النظام ويتيح للمستخدمين إصدار الأوامر إليه.

فالقشرة ليست سوى برنامج خدمة لإدخال الأوامر والوصول إلى نظام التشغيل, أي أنها في أغلب الأحيان لا تمثل جزءاً من **جوهر** نظام التشغيل.

يوجد لنظام يونيكس عدد من القشريات ، مثل Bourne,Bash، وKorn ، وC، و Bourne Again. ومستخدمو يونيكس يفضلون القشرية التي يختارونها، فيستغلون إمكانياتها الكامنة، ويضبطونها لتصبح مناسبة لبيئات عملهم، وينشئون الأسماء المستعارة للأوامر التي يستخدمونها بكثرة، ويكتبون برامج لتنفيذ بعض أوامر النظام تلقائياً.

من مهام واجهة المستخدم:

- تزود المستخدمين بواجهة يتم التعامل من خلالها.
- تمكن من الوصول إلى خدمات النواة(kernel).
- تمكن من تشغيل التطبيقات أو البرامج.
- .إمكانية استعراض محتويات الأدلة من خلالها.

وهناك عدة أنواع وأشكال من واجهات المستخدم, منها :

أولاً: واجهة الأوامر النصية (Command Line Interface-CLI):

واجهة الأوامر النصية هي طريقة يتم فيها تفاعل المستخدم مع البرامج أو نظام التشغيل باستخدام الأوامر الخطية بحيث يستجيب المستخدم مع رسائل الكمبيوتر التوجيهية من خلال طباعة الأوامر ومن ثم يتلقى المستخدم إجابة من النظام. ويستخدمها عادة المبرمجون ومدراء الأنظمة والأشخاص ذوي الخبرة التقنية.

---

[3]إعداد: بشائر الخويطر، أنفال العواجي، قطرالندى عبد الرحمن السماعيل

تحتاج هذه الواجهة إلى مترجم يسمى command line interpreter , وهو برنامج يقرأ الأوامر الخطية المدخلة من المستخدم ويترجمها في سياق نظام التشغيل أو لغة البرمجة المستخدمة.

من أمثلة هذه الواجهة:

MS-DOS command line interface وهي المستخدمة في نظام الويندوز.

ثانياً: واجهة المستخدم الرسومية (Graphical User Interface- GUI):

واجهة المستخدم الرسومية تؤمن التفاعل مع الحاسب باستخدام أغراض و صور رسومية –**أيقونات**– وهي غالباً تتكون من عناصر تحكم. إضافة إلى نصوص توجه المستخدم لاستخدام أحداث مخصصة مثل نقر الفأرة, أو توجهه إلى إدخال نصوص ليقوم الحاسب بما يريده المستخدم . جميع الأفعال و المهام التي يمكن للحاسب تنفيذها تتم عن طريق التطبيق المباشر لأحداث على العناصر الرسومية (عناصر التحكم).

أكثر المستخدمين اليوم يفضلون الواجهة الرسومية على واجهة الأوامر الخطية ،كما أن أغلب أنظمة التشغيل اليوم توفر كلا الواجهتين للمستخدم.

ومن الأمثلة على الأنواع الأخرى لواجهة المستخدم:

الواجهة الرسومية القابلة للتكبير (ZUI - zoomable user interface):

هي نوع من أنواع الواجهات الرسومية , ولكنها تختلف عن الواجهة العادية في أنها لا تستخدم النوافذ, حيث أن العناصر تظهر على سطح المكتب, وإذا تم اختيار العنصر فإنه بدلاً من أن يُفتح في نافذة فإنه يتم تكبيره إلى المستوى المطلوب والعمل عليه, وعند الانتهاء يتم تصغيره على سطح المكتب .هناك عدة تطبيقات تستخدم هذا النوع منها: Google Earth , Google Maps, iPhone

المصادر:

- Operating system Concepts (Seventh Edition): Abraham Silberschatz,Peter Baer Galvin,Greg Gagne
- http://ar.wikipedia.org/wiki
- http://en.wikipedia.org/wiki
- http://www.opendirectorysite.info
- الموسوعة العربية للكمبيوتر والانترنت
- العربية مواقع الكتب
- http://en.wikipedia.org/wiki/ZUI

ثالثاً: نداءات النظام(System Calls)[4]

تعريفها:

نداءات النظام هي ميكانيكية تستخدمها برامج التطبيقات للحصول على خدمة يقوم بها نظام التشغيل. أو هي الطريقة التي يستخدمها (عملية المستخدم) ليسأل نظام التشغيل لفعل شيء معين.

متى تحدث ؟

تحدث نداءات النظام في وقت معالجة برامج التشغيل في الذاكرة حيث تحتاج إلى خدمات نظام التشغيل, مثل استخدام الأجهزة الملحقة بالنظام كبطاقة الشبكة أو بطاقة الصوت أو بطاقة الرسومات أو في الاتصالات بين البرامج التطبيقية.

عندما تُستخدم نداءات النظام, فإنه ببساطة يتم تحديد اسم الدالة المطلوبة ومناداتها, ولكن ما العمل عند الحاجة إلى معلومات إضافية؟ في هذه الحالة يتم إرسال هذه البيانات الإضافية عن طريق معاملات(parameters). وهناك عدة طرق تستخدم لإرسال المعاملات وهي:

1. إرسال المعاملات إلى النظام عن طريق وضعها في أحد المسجلات(register). هذه الطريقة سريعة ولكنها تُفضل فقط عندما يكون لدينا عدد قليل من المعاملات وذلك لأنه هناك عدد محدود من المسجلات داخل المعالج.

2. استخدام تركيب بيانات من نوع stack , بحيث يقوم البرنامج بدفع المعاملات داخلها ومن ثم يقوم نظام التشغيل باستخراجها. هذه الطريقة لا تحددنا في كمية البيانات المخزنة.

3. تخزين المعاملات في مكان محدد في الذاكرة(block) أو توضع في جدول في الذاكرة(table), ثم بعد ذلك يوضع عنوان المكان أو الجدول في مسجل(register) ويمرر هذا العنوان إلى نظام التشغيل. نستطيع القول أن هذه الطريقة تعتبر من أفضل الطرق الثلاث وذلك لأنها لا تحددنا في كمية المعلومات المخزنة , بالإضافة إلى أن النظام يستطيع الوصول إلى أي معلومة بسهولة على

---

[4] إعداد: منى البريه، خديجة أخرفي، قطر الندى عبدالرحمن السماعيل، نهى الطياش، نوف السفياني

عكس الطريقة السابقة, حيث إذا أراد النظام معلومة في أسفل الــstack فسوف يضطر لإخراج جميع المعلومات التي تقع فوقها!

يمكن تصنيف نداءات النظام إلى هذه الأنواع:

1. **أعمال الملفات:** خلق / حذف/ فتح ملف ، قراءة / كتابة
2. **إدارة الأجهزة** : طلب / تحرير ، قراءة / كتابة
3. **صيانة المعلومات** : طلب /أخذ المعلومات ، معرفة الوقت و التاريخ وعملية الحصول على المعلومات
4. **التواصل** : خلق / حذف الروابط ، وإرسال / استقبال الرسائل
5. **التحكم في العمليات.**

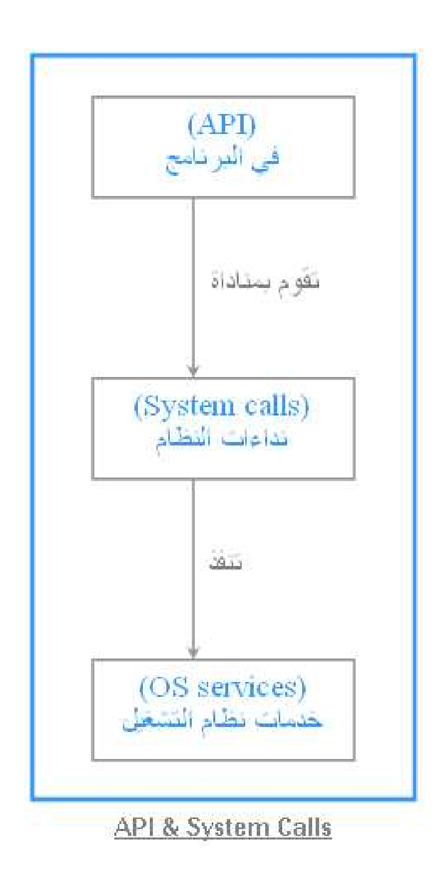
الأوامر البرمجية (API) application program interface :

لكل نظام تشغيل مجموعة من الأوامر البرمجية(API's), التي تقوم بمناداة نداءات النظام (system call) في قلب النظام (kernel mode) ثم تنتقل إلى نظام التشغيل. فمثلاً عند عمل الأمر البرمجي –**open()**– فإنه يستدعي نداء النظام لهذا الأمر:

(open system call>>>>> () open) .

وليس من الضروري أن يعادِل الأمر البرمجي الواحد نداء واحد للنظام! فقد يتطلب المئات من نداءات النظام.

الصورة التالية توضح لنا المفهوم العام لعلاقة الأوامر البرمجية مع نداءات النظام:



API & System Calls

من أكثر أنواع الأوامر البرمجية شيوعاً:

1. Win32 API التي تستخدم في نظام الويندوز
2. POSIX API التي تستخدم في أنظمة UNIX و Linux و Mac OS X
3. JAVA API المستخدمة في الآلة الوهمية للغة الجافا.

إن من الجدير بالذكر أنه يُفَضَّل استخدام الأوامر البرمجية بدلاً من نداءات النظام, وذلك للأسباب التالية:

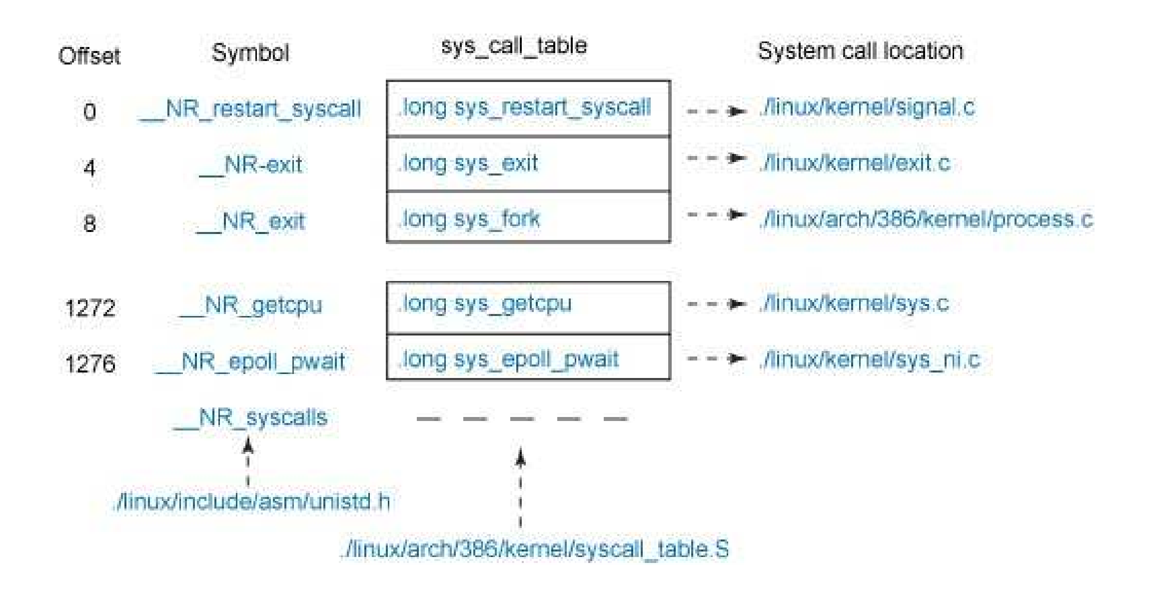
1. قابلية النقل(Portability) في الأوامر البرمجية.

فمثلا: عند كتابة برنامج بلغة الجافا فإننا نستطيع تشغيله في أي نظام تشغيل مثل Windows وLinux وMac بدون أي تغيير في أوامر نداء النظام.

2. أوامر استدعاء النظام أكثر تفصيلاً وتعقيداً ويصعب التعامل معها وتختلف من نظام لنظام, بينما الأوامر البرمجية أسهل وأقل تعقيداً.

الآن نعود إلى كيفية تنفيذ نداءات النظام داخل نظام التشغيل (system calls implementation of):

يتم ربط الأمر البرمجي (API) برقم (index) وهذا الرقم يُربط بأمر نداء النظام , وتلك الأرقام تكون مدونة في جدول يسمى جدول النظام(system table) , ولكل نداء للنظام رقم (index) .
الشكل التالي يوضح جدول النظام حيث نلاحظ فيه كل أمر نظام مرتبط برقم(index):



مثلاً عند عمل الأمر البرمجي –**open()**– , سوف يتم الانتقال من أسلوب المستخدم إلى أسلوب لب النظام, وبالتالي لابد من استخدام نداء النظام, ويتم ذلك بعمل مناداة للرقم المرتبط بالنداء الذي يتم إيجاده من خلال جدول النظام (system table) كما هو موضح في الشكل التالي:

المصادر:

- Operating system Concepts (Seventh Edition): Abraham Silberschatz,Peter Baer Galvin,Greg Gagne
- http://en.wikipedia.org/wiki/System_call
- http://data.uta.edu/~ramesh/cse3320
- http://tiger.la.asu.edu/Quick_Ref/Linux_Syscall_quickref.pdf
- http://www.slideshare.net/guestd1b5cb/adding-a-system-call

رابعاً: برامج النظام[5](System Programs)

هي مجموعة برامج توفر بيئة تخاطبية بين نظام التشغيل والبرامج المطوَّرة من قبل المستخدمين ومطوري البرامج, وأكثر المستخدمين يتعاملون مع نظام التشغيل عن طريق برامج النظام وليس عن طريق الاتصال المباشر بنظم التشغيل.

أنواع برامج النظام:

تقسم برامج النظام إلى عدة أقسام وهي:

- **إدارة الملفات:** وهي المسئولة عن خلق, حذف, إعادة تسمية, نسخ وغيرها من العمليات على الملفات والأدلة.
- **معلومات حالة النظام:** هي برامج تسأل النظام عن الوقت, التاريخ, حجم الذاكرة, عدد المستخدمين.
- **تعديل الملفات:** وهي عبارة عن مجموعة من محررات النصوص لعمل تغيرات في محتويات الملفات.
- **دعم ملفات البرمجة:** وهي المسئولة عن التجميع في برامج لغات البرمجة.
- **تنفيذ وتحميل البرامج:** وهي المسئولة عن تنفيذ البرامج بعد تحميلها.
- **الاتصالات:** وهي المسئولة عن التواصل بين العمليات أو بين المستخدمين أو بين أجهزة أخرى مختلفة.

المصدر:

- Operating system Concepts (Seventh Edition): Abraham Silberschatz,Peter Baer Galvin,Greg Gagne

---

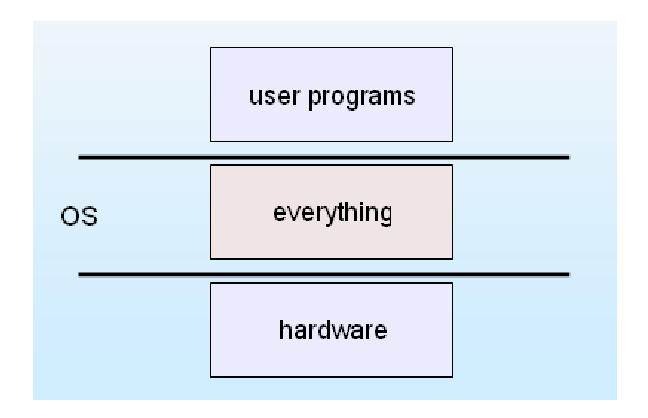
[5] إعداد: أمل الصبيح

خامساً: تركيب نظم التشغيل(**Operating Systems Structure**)[6]

هناك عدة طرق لبناء وتركيب نظم التشغيل و هي:

1. التركيب البسيط (**Monolithic**) :

بحيث يكون نظام التشغيل في مستوى واحد أو في مستويين.



الميزة الرئيسية : تكلفة التفاعلات الداخلية في النظام تكون منخفضة لأنها جميعا تقع على نفس المستوى.

العيوب:

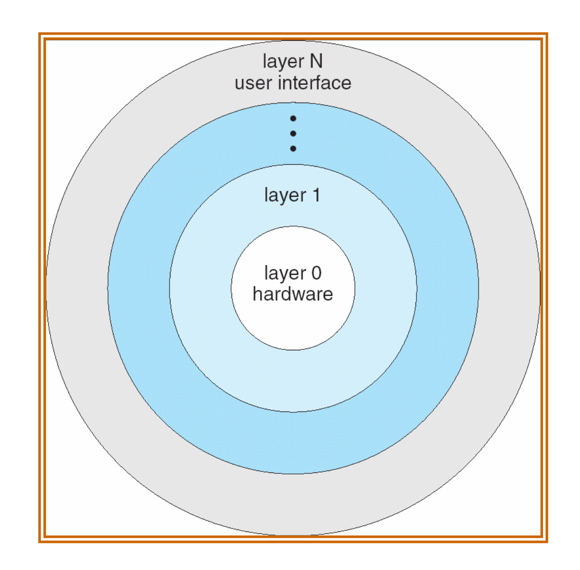
- صعوبة الفهم .
- صعوبة التعديل .
- صعوبة الصيانة .
- غير موثوق فيه .

2. تركيب الطبقات (**Layered**) :

أي أن نظام التشغيل مقسم لطبقات (مستويات) بحيث يكون كل جزء من النظام في طبقة مستقلة ، و بحيث أن الطبقة 0 (layer 0) مخصصة للعتاد (Hardware) ، والطبقة **ن** (layer N) مخصصة لواجهة المستخدم, كما هو موضح في الشكل التالي:

---

[6] إعداد: سمية الخنيزان



الميزة الرئيسية : وجود الطبقات أدّى إلى تسهيل عملية الصيانة .

العيوب: المشكلة تكمن في عملية ترتيب الطبقات ، فلا توجد لدينا طريقة واضحة للترتيب .

3. تركيب النواة الصغيرة (**Microkernel**) :

تكون نواة النظام في هذا التركيب صغيرة جداً ، ولا يوضع بداخلها سوى الوظائف الأساسية . أما الوظائف الأخرى فتوضع في مساحة المستخدم ، ويكون الاتصال بين مساحة المستخدم والـــنواة عن طريق الـــرسائل العابرة (message passing).

الميزات :

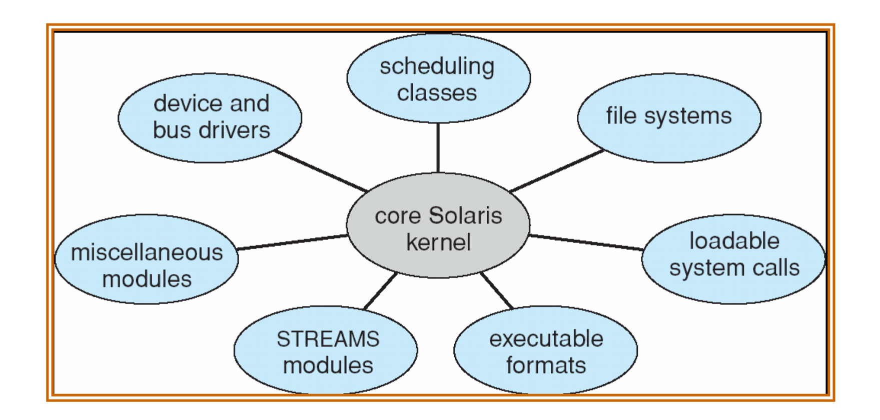
- من السهل توسيع (تمديد) النظام.
- النظام أكثر ثقة و أكثر أمناً.

العيوب :

الاتصال بين مساحة المستخدم ونواة النظام عملية مكلفة.

4. تركيب الوحدات (**Modules-based**) :

معظم أنظمة التشغيل الحديثة مبنية بهذه الطريقة ، حيث تكون النواة الأساسية في المركز وبقية الوظائف تتفرع منها ، وهي مشابهة للطبقات ولكن أكثر مرونة وأكثر كفاءة.



المصدر :

- Operating system Concepts (Seventh Edition): Abraham Silberschatz, Peter Baer Galvin, Greg Gagne

سادساً: الآلات الافتراضية أوالتخيلية(Virtual Machines[7])

ما هي الآلة الافتراضية؟

الآلة الافتراضية هي عبارة عن برنامج يسمح بتشغيل أكثر من نظام تشغيل تخيلي افتراضي على جهاز شخصي واحد. بحيث يمكن تثبيت أكثر من نظام تشغيل على نفس الجهاز والتنقل بين هذه الأنظمة دون المساس بالنظام الحالي ودون خسارة كبيرة في الأداء. و تعمل الآلة الافتراضية كتطبيق على نظام التشغيل المضيف.

خصائصها:

- تمكن من استخدام أنظمة تشغيل متعددة على جهاز واحد والتنقل بينها دون الحاجة لإعادة تشغيله .
- تمكن من تركيب أنظمة تشغيل متعددة دون تقسيم جديد للقرص الصلب.
- توفر الاتصال بين أنظمة تشغيل متعددة على جهاز شخصي واحد.

فوائـــدها:

- إمكانية تجربة أنظمة تشغيل متعددة بأقل التكاليف.
- إمكانية إجراء تعديلات وتجارب على النظام التخيلي بحرية وذلك لأن الموارد التي يستخدمها معزولة تماماً عن موارد النظام الأساسي.
- يساعد على توفير بيئة برمجية جيدة , مما يتيح لُمطوّري أنظمة التشغيل القيام بالتجارب والبحوث على الآلة الوهمية بدلاً من القيام بها على النظام الأساسي وبالتالي لا يؤثر على أداء هذا النظام.

مثال على الآلات الافتراضية:

آلة جافا الوهمية (JAVA Virtual Machine), التي تمكن ملفات الجافا من العمل على جميع أو معظم أنظمة التشغيل.

---

[7] إعداد: حليمة حكمي, نوف السفياني, نورة الخالدي

المصادر:

- http://www.fantookh.com/ طبيب الإنترنت الحلقة 122 بتاريخ 1421/3/29
- www.cis.nctu.edu.tw
- http://tarksiala.blogspot.com/2007/06/vmware-60.html

سابعاً : تنشئة نظام التشغيل(Operating system generation)[8]

من الممكن تصميم نظام تشغيل أو برمجته أو تنفيذه خصيصاً لجهاز واحد في مكان واحد, لكن نظم التشغيل بشكل عام مصممة للعمل على أي نوع من أجهزة الكمبيوتر في أي مكان ومتصلة مع أي نوع من الأجهزة الطرفية. لذلك يجب تركيب النظام لكل جهاز على حدة , وعملية التركيب هذه يطلق عليها (SYSTEM GENERATION) أي "تنشئة النظام".

في العادة تنتج الشركات نظم التشغيل على أقراص مدمجة (CD), ولكي تتم عملية تركيب النظام بشكل صحيح يجب استخدام برنامج يطلق عليه اسم "SYSGEN" الذي يقوم بقراءة بيانات التركيب من ملف معين, أو يقوم بتوجيه أسئلة إلى الشخص الذي يقوم بتركيب النظام حول معلومات تخص النظام , ومن هذه المعلومات على سبيل المثال:

- نوع المعالج المستخدم, و في حالة تعدد المعالجات يجب وصف كل معالج على حدة.
- حجم الذاكرة المتوفرة.
- الأجهزة المتوافرة, حيث يجب تحديد النوع والطراز وأي مواصفات خاصة.
- خيارات نظام التشغيل المرغوبة,مثل اختيار طريقة جدولة المعالج,أو العدد الأقصى من العمليات.

حالما يتم تحديد المعلومات فإنه يمكن حفظها في جداول وتخزينها, بحيث يتم استخدامها عند تركيب النظام على أجهزة أخرى, وبالتالي توفر على المستخدم عناء تكرار إدخال جميع هذه المعلومات على كل جهاز في كل مرة.

المصدر:

- Operating system Concepts (Seventh Edition): Abraham Silberschatz, Peter Baer Galvin, Greg Gagne

[8] إعداد: سمية باعطوة

الفصل الثالث:

العمليات

العمليات

Processes

مقدمة

في بدايات الحاسب كان النظام يسمح لبرنامج واحد أن ينفذ في وقت معين. وكان هذا البرنامج يسيطر سيطرة تامة على النظام.

ولكن في الحاسبات الحالية يسمح النظام لأكثر من برنامج أن يحمّل إلى الذاكرة وأن ينفذون في نفس الوقت. وهذا التطور يتطلب تحكم أكبر وتقسيم البرامج المختلفة إلى أجزاء مستقلة، وهذه الاحتياجات أنتجت لنا ما يدعى بالعملية process، وهي البرنامج في مرحلة التنفيذ. والعملية هي وحدة العمل في أنظمة مشاركة الوقت time-sharing الحديثة.

وكلما كان نظام التشغيل معقداً، كلما توقعنا منه عمل أموراً أكثر بالنيابة عن مستخدميه. لذا يتكون النظام من مجموعة من العمليات: عمليات نظام التشغيل تنفذ شفرة (code) النظام، وعمليات المستخدم تنفذ شفرة المستخدم. ومن الممكن أن تعمل كل هذه العمليات في نفس الوقت، بجعل وحدة المعالجة المركزية (CPU) تعمل عليهم بتعدد multiplexed. بتبديل وحدة المعالجة بين العمليات، يمكن أن يجعل نظام التشغيل الحاسب أكثر إنتاجيه.

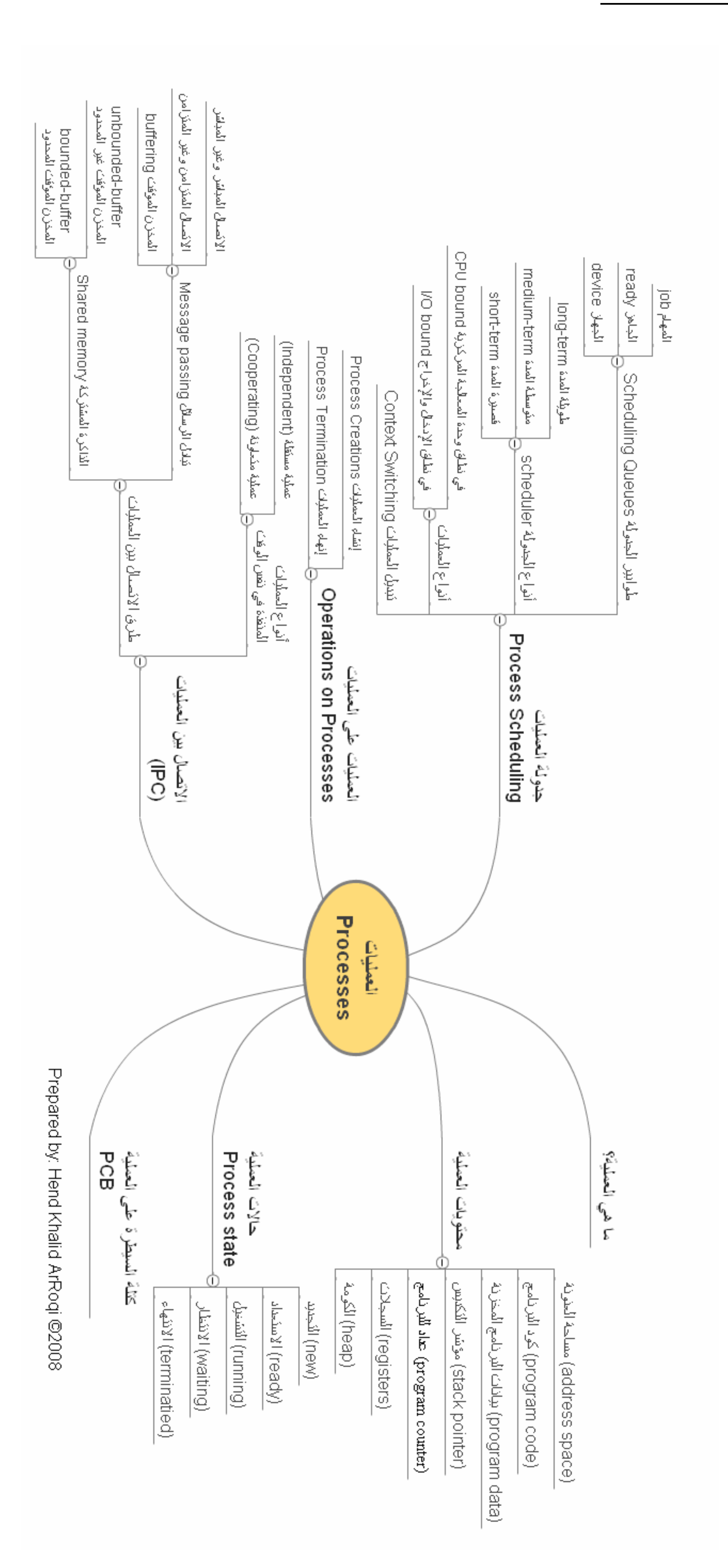
هذه كانت بداية تعريفية عن العمليات كان المرجع فيها كتاب:

Operating System Concepts: Silbreschatz, Galvin and Gagne, 7th edition وفيما يلي شرح مفصل عن بعض المواضيع المتعلقة بالعمليات في نظم التشغيل، تم جمعها من ويكي wiki مادة نظم التشغيل:

- http://os1h.pbwiki.com/
- http://os2h.pbwiki.com/
- http://os3h.pbwiki.com/
- http://os2a.pbwiki.com/

وإعدادها وتنسيقها بواسطة هند خالد الروقي (Hend.khalid@gmail.com)

الخريطة الذهنية للفصل:



أولاً: مفهوم العملية Process Concept[9]

العملية هي تنفيذ برنامج متسلسل، وعموما يحتاج تنفيذ العملية إلى عدة موارد منها: وقت وحدة المعالجة المركزية (CPU), الذاكرة، الملفات وأجهزة الإدخال والإخراج. وهذه الموارد تعطى للمهمة إما وقت إنشاءها أو وقت تنفيذها.

والعملية عادة هي وحدة عمل متكاملة في أغلب أنظمة التشغيل (operating systems)، وهي إما عمليات خاصة بأنظمة التشغيل وتنفذ برامج التشغيل، أو عمليات خاصة بالمستخدمين وتنفذ برامج المستخدمين. وجميع هذه العمليات تنفذ في نفس الوقت.

والمسؤول عن إدارة هذه العمليات وكل ما يتعلق بها من إنشاء أو إلغاء وجدولة وآلية تزامن واتصالات العمليات هو نظام التشغيل.

العملية Process[10]

تحتوي العمليات (على الأقل) على:

- مساحه العنونة (address space) هي مساحه محجوزة بالذاكرة (تحتوي على معلومات العملية)
- الكود المستخدم في البرنامج المراد تنفيذه (program code).
- البيانات المخزنة للبرنامج المراد تنفيذه (program data).
- ومؤشر للتكديس (stack pointer).
- عداد للبرنامج (program counter) (حيث أن البرنامج يتألف من عدة سطور وهذا العداد يعد هذه الأسطر).
- السجل (register) وقيمه.
- الكومة (heap) عمليه تطويق أو تحديد البيانات التي استخدمناها في هذه العملية.

---

[9] نهى الطياش

المصدر: http://en.wikipedia.org/wiki/Process_%28computing%29

[10] آلاء العباد

ثانياً: حالات العمليات Processes State[11]

كل عمليه من العمليات لابد أن تمر بأكثر من حاله وقت تنفيذها, هذه الحالات تدل على نشاطها في هذه اللحظة.

الحالات التي تمر بها أي عمليه هي:

التجديد (new) (أو حالة لم تستخدم من قبل أو بالأصح لم يُفعَل استخدامها):

وهي وقت تعريف العملية ووقت السماح لها بالدخول إلى قائمه العمليات الموجودة في الذاكرة الرئيسية RAM ويتم ذلك بالضغط على البرنامج ضغطة مزدوجة وبالتالي تنتقل هذه الحالة من الحالة الخاملة إلى حاله أخرى، مثل:"حاله التنشيط".

الاستعداد (ready):

هي العملية الجاهزة للتنفيذ والدخول إلى وحدة المعالجة المركزيةCPU ، ولن يسمح لها بالتنفيذ بسبب وجود عمليه أخرى تنفذ في نفس الوقت.

التشغيل أو التنفيذ (Running):

هي حالة العمليات والأوامر وقت التنفيذ في وحدة المعالجة المركزيةCPU .

الانتظار(waiting):

هي حالة العملية عند انتظار حدوث أمر معين، مثلا: ينظر إدخال بيانات من المستخدم أو عمليه طباعة.

الانتهاء(terminated):

---

[11] لمياء الجاسر

المرجع: Operating System Concepts: Silbreschatz, Galvin and Gagne, 7th edition

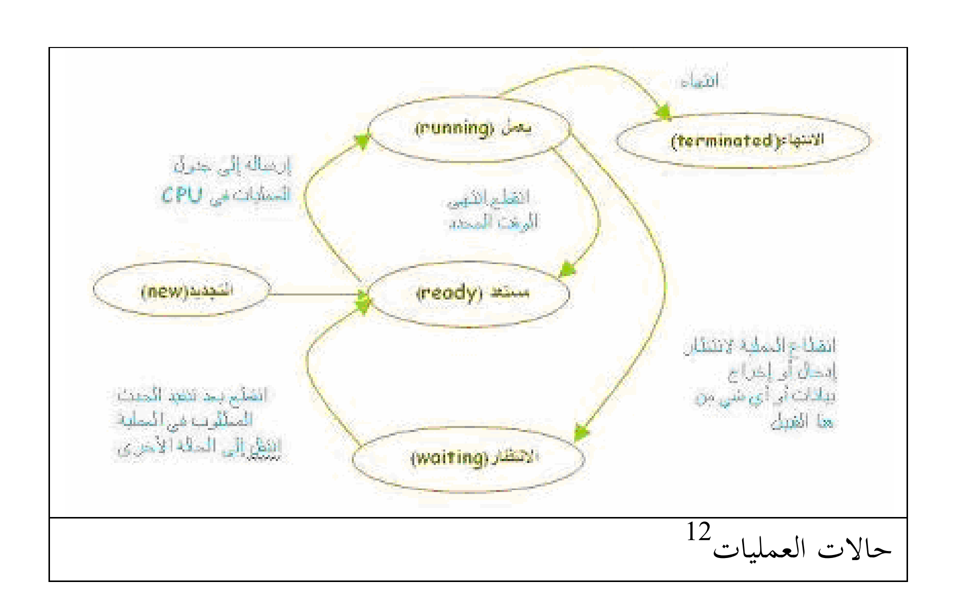
هي حالة العملية عند الانتهاء, وهي إما أن تكون العملية قد انتهت بشكل سليم أو أنه قد حصل لها خطأ معين أدى إلى إنهاءها.

ومن مفهومنا للحالات يمكن أن نستنتج كم عملية يمكن أن تتم في كل حالة.

-في حالة (الاستعداد): يمكن أن توجد أكثر من عملية في حالة استعداد في نفس الوقت، وكل هذه العمليات مستعدة للتنفيذ في أي وقت.

-في حالة (التشغيل): فإنه في وقت معين يتم تشغيل عملية واحدة على وحدة المعالجة المركزية (CPU)، ولا يمكن أن يكون هناك أكثر من عملية تعمل على وحدة المعالجة المركزية (CPU) في نفس الوقت (أي في حالة التشغيل) ، فقط عملية واحدة تنفذ في وقت واحد ولا يمكن أن ينفذ أكثر من ذلك.

-في حالة (الانتظار): تشبه حالة (الاستعداد) من الممكن أن تكون هناك أكثر من عملية في حالة انتظار لحدث معين أياً كان هذا الحدث في نفس الوقت.



حالات العمليات[12]

شرح مبسط للرسم الموضح أعلاه:

[12] من حقوق لمياء الجاسر

تبدأ العملية من حاله التجديد وذلك بالضغط عليها من جهاز الحاسب لديك ثم تنتقل بعد ذلك إلى حالة (الاستعداد) وهذه الحالة يتم إضافتها إلى الجدولة في (CPU) ليتم تنفيذها عندما يحين الوقت المخصص لها

تبدأ العملية بالتنفيذ وتنتقل من حالة إلى حالة في حالات معينة:

- تنتقل إلى حالة الانتهاء(terminated) عندما تنتهي العملية بسلام بشكل كامل أو عند حدوث خطأ معين أدى إلى أن يقرر النظام إنهاء العملية
- تنتقل إلى حالة الاستعداد (ready) عندما ينتهي الوقت المحدد لهذه العملية ولا تحتاج إلى تنفيذ حدث معين سواء إدخال بيانات أو غيره
- تنتقل إلى حالة الانتظار(waiting) عندما تكون العملية تمت بشكل جزئي ولكن تحتاج إلى حدث معين يطلب من المستخدم سواء إدخال أو طباعة أو أوامر أخرى
- عندما تكون العملية في حالة الانتظار وانتهى الحدث المطلوب تنتقل من حالتها إلى حاله الاستعداد، إذا انتهى الحدث بشكل كامل فهي الآن مستعدة للتنفيذ.

كتلة السيطرة على العملية Process Control Block[13]

كل عملية تمثل في نظام التشغيل بكتلة السيطرة على العملية (Process Control Block) ويرمز لها بالرمز PCB وهي تراكيب بيانات في نواة نظام التشغيل تحتوي على المعلومات اللازمة لإدارة عملية معينة, ويختلف تنفيذها من نظام لآخر, ولكن بشكل عام ستشمل ما يلي بشكل مباشر أو غير مباشر :

- حالة العملية (state): يمكن أن تكون جديدة, جاهزة, قيد التشغيل, في حالة انتظار أو تم إيقافها.

[13] هند المطيري
المرجع: Operating System Concepts: Silbreschatz, Galvin and Gagne, 7th edition
http://en.wikipedia.org/wiki/Process_control_block

- عداد البرنامج (Program Counter): يشير العداد إلى عنوان الأمر القادم الذي سينفذ في هذه العملية.

- سجلات وحدة المعالجة المركزية (Registers): تتفاوت السجلات في العدد والنوع اعتمادا على هندسة الحاسوب, وتحتوي على مؤشرات للكومة (للمكدّس) (stack pointer), سجلات الفهرسة (index registers)، سجلات متعددة الأغراض (general-purpose registers) بالإضافة إلى أي معلومات شرطية في الكود (condition-code information) ، قيم السجلات يجب أن تحفظ عند حدوث مقاطعة للعملية, كي تسمح للعملية أن تستمر بشكل صحيح عندما يتم تشغيلها لاحقاً.

- معلومات جدولة وحدة المعالجة المركزية (CPU-scheduling information): تتضمن أولوية العملية, مؤشرات على صفوف الجدولة وأي عوامل خاصة بالجدولة.

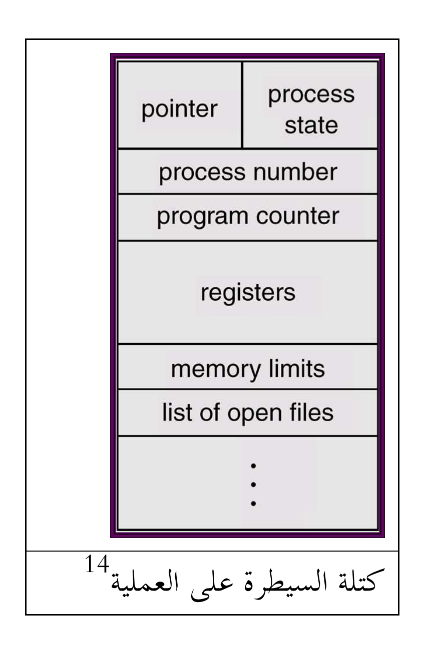
- معلومات إدارة الذاكرة (memory-management information): وهي تحتوي على معلومات عن قيم سجلات البداية (base) والنهاية (limit), وجداول الأقسام (segment table) وجداول الصفحات (page table) وذلك اعتماداً على نظام الذاكرة المستخدم من قبل نظام التشغيل.

- المعلومات الحسابية للعملية (Accounting information): تتضمن كمية وحدة المعالجة المركزية والوقت الحقيقي اللذان تم استخدامهما من قبل العملية.

- معلومات عن حالة الإدخال والإخراج (I/O state information): تتضمن قائمة أجهزة الإدخال والإخراج التي خصصت للعملية, قائمة الملفات المفتوحة وإلى آخره.

- مؤشر على العملية التالية التي يجب تنفيذها، أي مؤشر على PCB للعملية التالية.

أثناء التبديل إلى عملية أخرى, يتم إيقاف العملية الحالية أي التي تكون قيد التشغيل وتشغيل العملية الأخرى. في هذه الحالة يجب أن تعمل النواة على إيقاف العملية الحالية وتعطي نسخة من قيم السجلات لكتلة السيطرة على العملية (PCB) الخاصة بهذه العملية, ثم تجدد قيم السجلات بقيم كتلة السيطرة على العملية (PCB) للعملية الجديدة.



كتلة السيطرة على العملية[14]

موقع كتلة السيطرة على العملية (PCB):

بما أن كتلة السيطرة على العملية تحوي معلومات حساسة ومهمة فإنها يجب أن توضع في منطقة الذاكرة المحمية من وصول المستخدم العادي.

في بعض أنظمة التشغيل يتم وضعها في بداية كومة النواة (kernel stack) للعملية لأنه موقع محمي ومناسب.

ثالثاً: جدولة العمليات Process Scheduling [15]

---

[14] www.ice.ntnu.edu.tw/~swanky/os/chap4.htm

[15] نوره المحيسن المراجع: Operating System Concepts: Silbreschatz, Galvin and Gagne, 7th edition

http://en.wikipedia.org/wiki/Scheduling

من المنطقي جدا أن يكون هناك طريقة لجمع المهام في مكان واحد بشكل منظم ومرتب هذا المكان يسمى بالطابور (queue)

طوابير الجدولة Scheduling Queues:

1. طابور المهام (job queue): يوجد فيه جميع العمليات الموجودة في النظام

2. الطابور الجاهز (ready queue): به جميع المهام التي تنتظر التنفيذ

3. طابور الجهاز (device queue): جميع المهام التي تنتظر مدخلات أو مخرجات.[16]

جدولة العمليات:

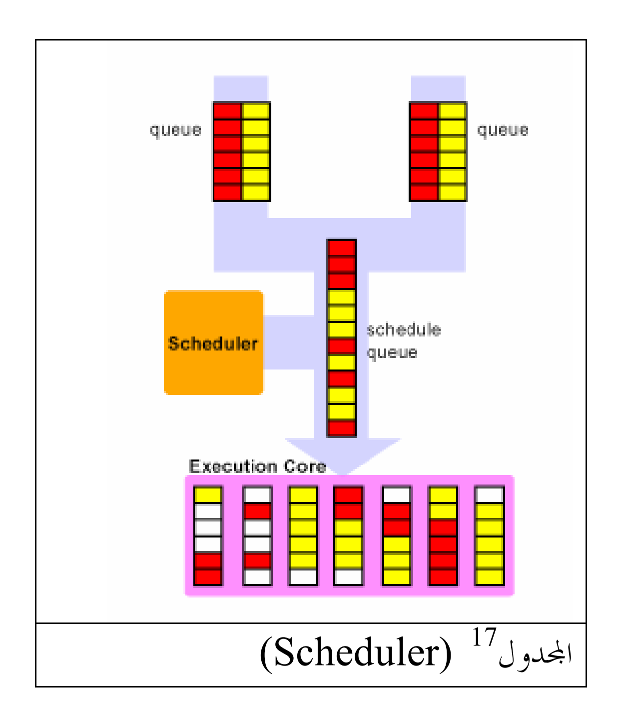
هي وضع خطة لترتيب دخول العمليات على المعالج بحيث تدخل عملية واحده كل مرة ونستغل معظم وقت المعالج و يقوم (مجدول العمليات) بترتيب دخول العمليات على المعالج

---

http://en.wikipedia.org/wiki/Scheduling_%28computing%29

http://en.wikipedia.org/wiki/Ready_queue

[16] فايزة المطيري
المرجع: Operating System Concepts: Silbreschatz, Galvin and Gagne, 7th edition



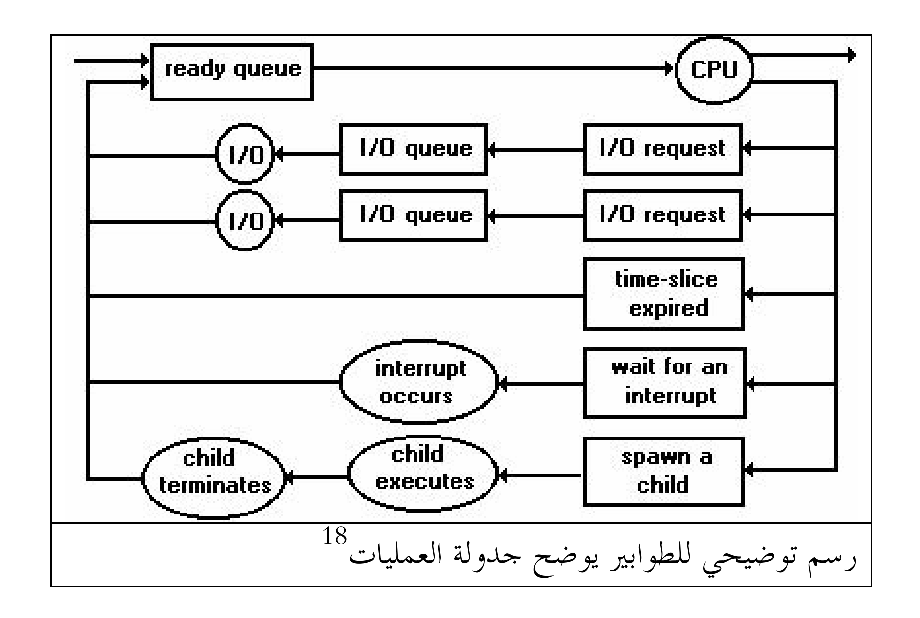
(Scheduler) المجدول[17]

جدولة الطوابير queue scheduling:

عندما تدخل عملية إلى النظام فإنها تدخل في طابور المهام (job queue) الذي يحتوي جميع عمليات النظام وعندما تصبح العملية جاهزة و تنتظر التنفيذ فإنها تنتقل إلى الطابور الجاهز(ready queue)

أما إذا كانت العملية تنتظر عملية إدخال أو إخراج مثل التحميل من القرص الصلب أو كانت تخدم اتصال انترنت فإنها تنتقل إلى طابور الإدخال والإخراج (I/O queue)

[17] http://arstechnica.com/articles/paedia/cpu/hyperthreading.ars/4



رسم توضيحي للطوابير يوضح جدولة العمليات[18]

أنواع الجدولة scheduler:

الجدولة طويلة المدى long-term scheduler:

و هي التي تقرر أي العمليات ستدخل إلى الطابور الجاهز و أيها تخرج أو تتأخر، وهذه الجدولة ليست موجودة في الحاسبات المكتبية فالعمليات تدخل إلى المعالج آلياً ولكنها مهمة لنظام الوقت الحقيقي (real time system) والالتزام بمواعيد العمليات النهائية.

الجدولة متوسطة المدى medium term scheduler:

هذه الجدولة موجودة في كل الأنظمة ذات الذاكرة الافتراضية (virtual memory)، فهو يقوم بعملية التبديل أي أنه يزيل العمليات بشكل مؤقت من الذاكرة الرئيسية إلى الذاكرة الثانوية (secondary storage)، وذلك حسب أولوية العملية وما تحتاجه من مساحة على الذاكرة. في هذه الأيام معظم الأنظمة التي تدعم الانتقال من العنوان الافتراضي إلى العنوان الثانوي بدل التبديل بين الملفات تكون الجدولة متوسطة المدى فيها تؤدي دور الجدولة طويلة المدى.

الجدولة قصيرة المدى short-term scheduler:

[18] www.cs.wayne.edu/~tom/guide/os.html

تقرر أي العمليات الجاهزة سيتم معالجتها بعد إشارة المقاطعة أو بعد استدعاء النظام. وهي أسرع من الجدولة الطويلة أو المتوسطة حيث تأخذ القرارات في وقت قصير جداً، ويمكن أن تكون قادرة على إجبار العمليات على الخروج من المعالج و إدخال عمليات أخرى أو تسمح ببقاء العمليات في المعالج حتى تنتهي.

أنواع العمليات Types Of Processes[19]:

1. في نطاق وحدة المعالجة المركزية (CPU bound process)

تقضي هذه العملية معظم وقتها في الوحدة المعالجة المركزية (CPU)، وتكون فترات عملها على وحدة المعالجة المركزية طويلة (CPU burst).

2. في نطاق الإدخال والإخراج (I/O bound process)

تقضي هذه العملية معظم وقتها في الإدخال والإخراج (I/O)، وتكون فترات عملها في الإدخال والإخراج طويلة (I/O burst).

من المهم جداً للجدولة طويلة المدة أن تختار خليط جيد من العمليات في نطاق وحدة المعالجة والعمليات في نطاق الإدخال والإخراج، لأنه عندما تكون كل العمليات في نطاق الإدخال والإخراج فإن الطابور الجاهز(ready queue) سيكون خالياً تقريباً من أي عملية، وعندها لن يكون لدى الجدولة قصيرة المدة ما تفعله. وبالعكس، عندما تكون كل العمليات في نطاق وحدة المعالجة، فسوف يصبح طابور الإدخال والإخراج فارغ دائماً تقريباً[20]، وسوف تكون الأجهزة غير مستخدمة بالشكل المطلوب ويصبح النظام غير متوازن. لذا لكي يكون النظام ذو أداء أفضل يجب أن يمتلك خليط من العمليات في نطاق الإدخال والإخراج والعمليات في نطاق وحدة المعالجة

---

[19] ميّ العتيبي
المرجع: Operating System Concepts: Silbreschatz, Galvin and Gagne, 7th edition
[20] تقريباً: لأن العملية عندما تكون في نطاق وحدة المعالجة مثلاً فإن ذلك لا يعني عدم احتوائها على أي أمر إدخال أو إخراج، فبالتالي لا يمكننا أن نقول أن طابور الإدخال والإخراج سيصبح فارغاً تماماً، ولكن لن يوظف بالشكل المطلوب.

تبديل العمليات (context switching) [21]

تبديل العمليات: عبارة عن تبديل المعالج من عملية إلى أخرى أو من تشعب (thread) إلى آخر.
مجدول وحدة المعالجة (CPU scheduler) هو من يحدد متى يتم تنفيذ عمليه التبديل بين عمليتين.

البيئة (context) يتم التعبير عنها في كتلة السيطرة على العملية PCB لكل عملية، و يتضمن القيم الموجودة في سجلات المعالج (CPU registers)، حالة العملية، ومعلومات إدارة الذاكرة.

تتم آلية تبديل العمليات من خلال هذه المراحل:

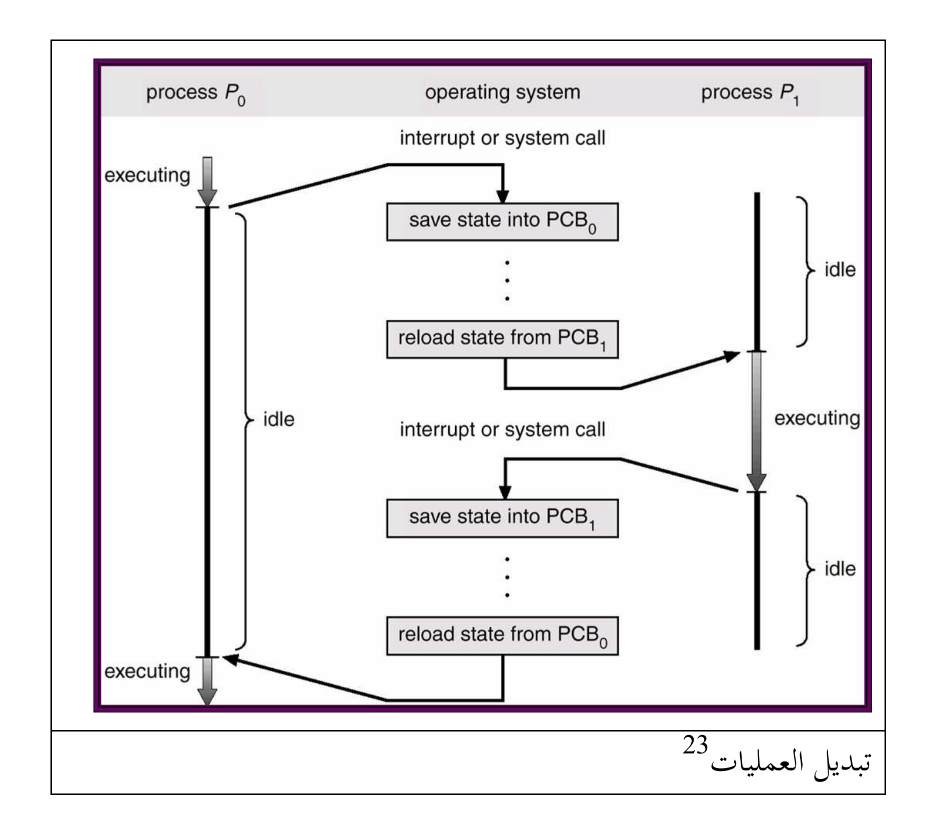
1. تأجيل إكمال عملية من العمليات وحفظ حالة المعالج لهذه العملية في مكان ما في الذاكرة وذلك عن حدوث قطع (interrupt) مثلاً.
2. إرجاع أو وضع بيئة (context) للعملية اللاحقة من الذاكرة وحفظها في سجلات المعالج.
3. الرجوع إلى المكان الذي يؤشر عليه عداد البرنامج(وهي العملية التي حصل عندها القطع (interrupt) وذلك لإكمال العملية.

خطوات تنفيذ عمليه التبديل بين عمليتين:[22]

1. ينتقل المعالج إلى النمط المميز (privileged mode)
2. نسخ محتوى السجلات (register) من العملية القديمة و تخزينها في جدول العملية
3. تحميل قيم السجلات (register) من جدول العملية الجديدة
4. يعود المعالج إلى نمط المستخدم (user mode)

---

[21] إيمان البلالي
المرجع: http://www.linfo.org/context_switch.html
[22] ديمه الثابت
المرجع: http://www2.cs.uregina.ca/~hamilton/courses/330/notes/processes/processes.html



تبديل العمليات[23]

العمليات على العمليات Operations on Processes

إنشاء العمليات Process Creations[24]

نظام العمليات يسمح بإنشاء العديد من العمليات الجديدة بواسطة استدعاء النظام (System Call) طيلة فترة التطبيق لهذه العملية، ويجب أن نعلم إنشاء العملية يطلق عليها (Parent Process) وهذا ما يقصد به الأب للعملية، والعملية الجديدة يطلق عليها (Child Process)، وهو الابن المنشأ من قبل العملية الأب.

---

[23] http://www.ice.ntnu.edu.tw/~swanky/os/chap4/CPU_Switch_From_Process_to_Process.png

[24] منار القحطاني
المرجع: Operating System Concepts: Silbreschatz, Galvin and Gagne, 7th edition

وكل تلك العلميات الجديدة (الأبناء) قد تستطيع إنشاء عمليات جديدة أخرى وبالإمكان تجميعها بالشكل الشجري للعمليات، قد تكون هناك أشياء مشتركة بين الأب والابن حيث أن الابن قد يكون نسخه طبق الأصل عن الأب، أو يشتركان في بعض الموارد، أو أن لا يكون بينهما أي موارد مشتركة.

وفي وقت التنفيذ إما أن ينفذا في وقت متزامن أو أن ينتظر الأب حتى تنتهي عمليات التطبيق الخاصة بالأبناء.

ويوجد أيضا احتمالات لمكان وجود العملية الجديدة (الابن) –الابن عملية مزدوجة من العملية الأم– الابن له برنامجه الخاص ومكان جديد يوجد به

والكثير من أنظمة التشغيل بما في ذلك اللينكس والويندوز تقوم بتعريف العمليات تبعا لمعرف العملية (Process Identifier) الخاص الذي يظهر عادة كرقم صحيح.

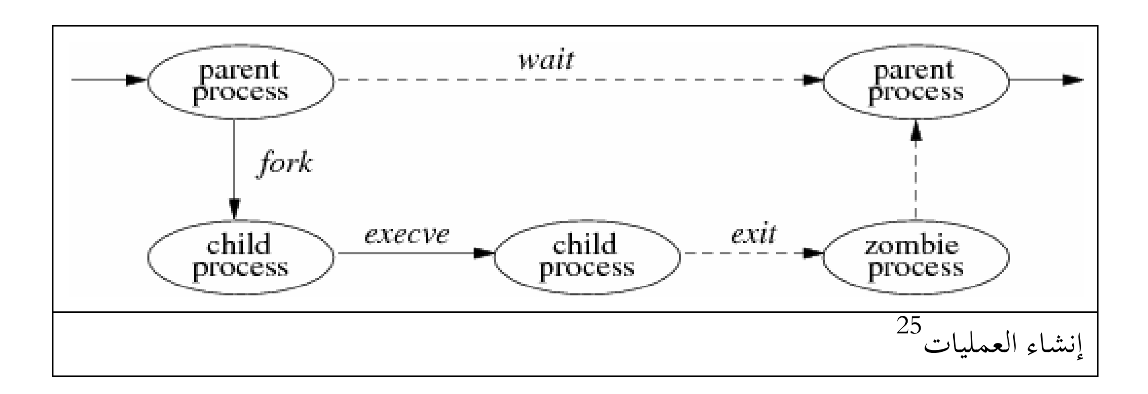
في نظام الينكس ، قائمة العمليات تستطيع أن تظهر بواسطة الأمر:

Ps command

حيث أنها تقوم بعرض كافة المعلومات لكل العمليات المطبقة في النظام.

وفي نظام اللينكس أيضا دالتين (function) في استدعاء النظام الأولى هي ما يطلق عليها fork() التي تعني انقسام للعملية وتكوين عملية جديدة حيث أنها تقوم بحجز مكان وذاكرة جديدة وعمل نسخة من ذاكرة الأب. عملية الانقسام هذه تنفذ مرتين في النظام أحداهما للأب والثانية للابن، وتقوم بإرجاع الرقم الخاص (PID) بالابن وتعطيه للأب وتقوم بإرجاع الرقم 0 للابن

والدالة الثانية هي دالة التطبيق exec() وهي التي تستخدم بعد دالة الانقسام السابقة لاستبدال ذاكرة العملية ببرنامج جديد تتخلص من المكان الحالي للعملية تحميل برنامج جديد للعملية الجديدة



إنشاء العمليات[25]

والجدير بالذكر أن دالة التطبيق لا تقوم بخلق عملية جديدة حيث أنها تعمل فقط على العمليات الحالية الموجودة.

إنهاء العمليات Process Termination[26]

أسباب إنهاء العمليات:

1. الخروج الطبيعي (Normal Exist): وتكون العملية في هذه الحالة قد أنهت عملها وتم إنهاءها.
2. الخروج بسبب خطأ (Error Exist): في هذه الحالة تكتشف العملية خطأ فادح (Fatal Error). مثال على ذلك محاولة تأليف (compile) لبرنامج غير موجود.
3. خطأ فادح (Fatal Error): وهنا يكون إنهاءها ناتج عن خطأ قامت به العملية، مثل: تنفيذها لأمر غير مسموح به كالقسمة على صفر أو الإحالة إلى مكان غير موجود في الذاكرة.
4. قتلها (kill) بواسطة عملية أخرى.

الاتصال بين العمليات Interprocess communication (IPC)[27]

هناك أنواع للعمليـــــــــات في نظام التشغيل خلال تنفيذ العملية:

---

[25] http://www.freebsd.org/doc/en_US.ISO8859-1/books/design-44bsd/book.html
[26] غيداءالفايز
المرجع:http://www.personal.kent.edu/~rmuhamma/OpSystems/Myos/processOperate.htm

[27] فاطمة الفرج, سجا الدرع , ريم الرشيدي
المرجع: Operating System Concepts: Silbreschatz, Galvin and Gagne, 7th edition

عملية مستقلة (Independent):

وهي العملية المستقلة التي لا تتأثر أو تأثر في تنفيذ عملية أخرى في النظام وإنما تعمل مستقلة بذاتها.

عملية متعاونة (Cooperating):

وهي العملية المتعاونة والتي يمكن أن تأثر أو تتأثر بتنفيذ عملية أخرى في النظام وهذه العملية تستخدم نوعين من الاتصال IPC:

1. تبادل الرسائل (Message passing):

حيث يتم تبادل الرسائل بين العمليات عن طريق الاتصال مع بعضها دون وجود مكان مشترك لتخزين الرسائل ويكون حجم هذه الرسائل ثابت أو متغير و تمر هذه الرسائل على لبّ النظام (Kernel) وتحتاج إلى اتصال بين العمليات.

2. الذاكرة المشتركة(Shared memory):

حيث يوجد بين العمليات مكان ذاكره مشتركة لكل منهم ويتم وضع الملفات المشتركة بداخلها.

مثلاً: العملية **أ** تنتج بيانات والعملية **ب** تريد أن تقرأ هذه البيانات من العملية **أ** فإن الذاكرة المشتركة تقوم بهذه المهمة لتسهل على العملية **ب** الوصول للمعلومات التي تريدها من العملية **أ** فهذه بيانات مشتركة يمكن لجميع العمليات الاستفادة منها عند وضعها في الذاكرة المشتركة.

مميزات العملية المتعاونة:

تقسيم المعلومات أو مشاركة الملفات.

تسريع انجاز العمليات.

تقسيم نظام المهام إلى عمليات منفصلة.

تمكن المستخدم من العمل على العديد من المهام في نفس الوقت.

نظم الذاكرة المشتركة Shared-Memory Systems:

(هي عبارة عن عملية مشاركة للذاكرة الافتراضية بين عمليتين أو أكثر, حتى يتم نقل وتبادل البيانات والاتصال بين كافة العمليات.)[28]

المشاكل التي تواجه المستخدمين والمنتجين بالنسبة للذاكرة المشتركة:
المخزن المؤقت غير المحدود (unbounded-buffer):
يمكن للمستخدم أن يقوم بعمليات غير محدودة فهي غير محدودة الحجم

المخزن المؤقت المحدود (bounded-buffer):
يكون هناك مساحة معينة للعمليات بقدر ما تنتج العملية الأولى من معلومات بقدر ما تستهلك العملية الثانية من هذه المعلومات فهي محدودة الحجم

ثانيا: الرسائل العابرة (massage- passing)[29]

وهي عبارة عن رسائل تتم بين العمليات لتبادل البيانات لكن قبل أن يتم الإرسال يجب أن يتم تحديد المرسل والمستقبل وأيضا يجب أن يكون بين الجهتين حلقة اتصال حتى يتم التبادل.

وهنا ثلاث طرق لتنفيذ وصلة اتصال منطقية وعمليات ارسل()/استقبل():

1. الاتصال المباشر أو غير المباشر.

---

[28] ابتهال العتيبي
المراجع: www.comms.scitech.susx.ac.uk/fft/computer/ipc.pdf
www.1006.org/os2006/labtext07_en.doc
http://www.alhasebat.com/vb/showthread.php?t=1878
http://www.cs.unc.edu/~dewan/242/f97/notes/ipc/node9.html
deneb.cs.kent.edu/~mikhail/classes/os.s00/L05ipcs.PDF

[29] ابتهال العتيبي
المراجع: www.comms.scitech.susx.ac.uk/fft/computer/ipc.pdf
www.1006.org/os2006/labtext07_en.doc
http://www.alhasebat.com/vb/showthread.php?t=1878
http://www.cs.unc.edu/~dewan/242/f97/notes/ipc/node9.html
deneb.cs.kent.edu/~mikhail/classes/os.s00/L05ipcs.PDF

2. الاتصال المتزامن أو غير المتزامن.

3. المخزن المؤقت (buffer)

1. الاتصال المباشر وغير المباشر:

المباشر: في هذا النوع من الاتصال نحتاج إلى معرفة المرسل ومعرفة المستقبل أي من الضروري تسمية الجهات حتى يتم الاتصال بينهم وفي هذا النوع لا نحتاج إلى مخزن مؤقت (buffer)لأن الاتصال يتم مباشرة من المرسل إلى المستقبل.

يتم الاتصال بهذه الطريقة:

send (receiver_process, message)

ارسل() : send()

اسم المستقبل هو : receiver_process

والرسالة المرسلة: message

وبنفس الطريقة للمستقبل من حيث نداء النظام system call:

receive (sender_process, message)

استقبل() : receive

اسم المرسل: sender_process

الرسالة المرسلة: message

غير المباشر: في هذا النوع لا نحتاج إلى اسم المستقبل أو المرسل, بل يتم إرسال الرسالة لصندوق البريد(mailbox)، حيث يتم إرسال الرسائل واستقبالها من صندوق البريد. الطريقة كالتالي:

(mailbox, message) Send

receive (mailbox, message)

وفي هذا الاتصال كل عملية لها صندوق بريد، وعادة يتم وضعه على شكل طابور (queue).

خصائص صندوق البريد:

1. لكل صندوق معرف خاص (unique identification)
2. وكل عملية تتصل مع الأخرى عن طريق صندوق البريد
3. أي عمليتين يتم الاتصال بينهما فقط إذا وجد صندوق بريد مشترك

خصائص وصلة الاتصال (communication link (CL:

1. يتكون الرابط بين عمليتين فقط إذا وجد صندوق بريد مشترك بينهن
2. من الممكن الربط بين أكثر من عمليتين
3. إذا وجد بين عمليتين أكثر من رابط, فإن كل رابط مخصص لصندوق بريد معين
4. الرابط ممكن أن يكون أحادي أو ثنائي الاتجاه

2. الاتصال المتزامن وغير المتزامن:

1. الاتصال المتزامن المحظور (blocking):

معنى الاتصال المتزامن المحظور أن الجهة المرسلة لا تستمر بإرسال الرسائل إلا عندما تصل الرسالة إلى الجهة المستقبلة, أي أنه تنتهي عملية الإرسال بوصول الرسالة إلى المستقبل وهذا الاتصال يعتبر آمن وأكثر بطئاً من غير المتزامن.

2. الاتصال المتزامن غير المحظور (unblocking):

الاتصال غير المتزامن غير المحظور (unblocking) تستمر الجهة المرسلة بالإرسال دون التأكد من وصولها للمستقبل, أي أنها ترسل الرسالة بطريقة موازاة (parallelism) ومن مميزاته السرعة لكنه أكثر خطورة من حيث الأخطاء بالنسبة للمتزامن.

3. المخزن المؤقت buffering:

لكـــل رابط بين العمليات سعة معينة لعدد من الرسائل التي تمر من خلاله.
السعة صفر zero : بمعنى أن الطابور (queue) طوله صفر أي لا يوجد أي رسائل به ومن المفترض هنا أن تكون العملية تزامنية بمعنى أن المرسل يجب أن ينتظر حتى تصل الرسالة إلى المستقبل وتسمى هذا الحالة الملتقى(rendezvous)

السعة محدودة : بمعنى أن الطابور (queue) يكون طوله n .
إذا كان الطابور للمستقبل غير ممتلئ, يمكن وضع رسالة جديدة فيه، والمرسل يستمر في إرسال الرسائل.
أما إذا كان الطابور ممتلئ المرسل يحظر الإرسال (block send) حتى يوجد مكان متاح للـــرسالة في الطابور.

السعة غير المحدودة: بمعنى أن الطابور يكون طوله إلى ما لا نهاية, وبهذه الحالة تستمر الجهة المرسلة بالإرسال دون توقف.

الاختلاف الأساسي بين حواسب تمرير الرسائل و المعالجات المشتركة بالذاكرة هو تنفيذ عمليات الاتصال على مستوى أنظمة الإدخال/الإخراج بدلاً من إجرائها في ذاكرة النظام.

	الذاكرة المشتركة	تبادل الرسائل
الأداء	جيد وفعال	جيد للتطبيقات التي لا تحتاج إلى اتصال كثيف بين المعالجات
التطبيق	سهلة التطبيق ولكنها مكلفة ماديا	سهلة التطبيق ومنخفضة التكلفة
وسيلة الاتصال	ذاكرة موحدة لجميع العمليات	عبر خطوط الاتصال في الشبكة
السلبيات	يجب توفير أساليب حماية الذاكرة المشتركة وتنظيم الوصول إليها	تكلفة تمرير الرسائل عالية جدا لأنها تتطلب عمليات دخل خرج كثيرة

جدول توضيحي يوضح الفرق بين نظام تبادل الرسائل ونظام الذاكرة المشتركة[30]

---

[30] فاطمة الفرج, سجا الدرع , ريم الرشيدي
المرجع: Operating System Concepts: Silbreschatz, Galvin and Gagne, 7th edition

الفصل الرابع:
الخيوط

التشعبات (الخيوط)
Threads

مقدمة:

مفهوم العملية (وأحيانا يطلق عليها مهمة) ومفهوم الخيط متماثلين وكثيرا ما يتم الخلط بينهما , فمعظم أجهزة الحاسب لا يمكن أن تنفذ إلا أمرا واحدا من برنامج معين , ولكن بما أنها تعمل بسرعة كبيره بالتالي تستطيع تشغيل العديد من البرامج ليخدم الكثير من المستخدمين في وقت واحد. تنقسم العملية إلى مجموعه من المهام (الخيوط أو ما تسمى بالتشعبات) بحيث يتم تنفيذ هذه المهام بشكل متزامن.
توجد الخيوط داخل كل عملية، وعندما تتم مناداة برنامج ما فإن نظام التشغيل يقوم بخلق عملية جديدة وكل عملية بدورها تقوم بخلق خيط لتنفيذ وإدارة العملية , وبإمكان كل خيط أن يخلق خيوط إضافية وجميع هذه الخيوط تشغل البرنامج نفسه بالعملية نفسها , و لكن كل خيط يقوم بتنفيذ جزء من البرنامج في أي وقت معين.
الخيوط تتم عن طريق وضع آليه تسمح للبرنامج بان ينفذ اكثر من امر في نفس الوقت , بحيث تقوم نواة اللينكس بتكوين جداول للخيوط بشكل غير متزامن بحيث تتم مقاطعة الخيوط من وقت لآخر حتى تعطي الآخرين فرصه للتنفيذ وذالك في حالة اجهزة الكمبيوتر التي تحتوي على معالج واحد فقط اما في حالة الأجهزه التي تحتوي على اكثر من معالج فيتم تنفيذ الخيوط بشكل متوازي .

المفاهيم التي سيتم دراستها :

1- المهمة وأنواعها .
2- الخيوط (التشعبات).
3- مكتبة سلسلة المهام .
4- تجزئة المهمة .
5- ربط الخيوط في مساحة المستخدم إلى مساحة النواة .
6- خيوط مستوى النواة , خيوط مستوى المستخدم

7- أمر التفرع والتنفيذ .

8- إلغاء تفرع الخيوط .

9- حامل الإشارة .

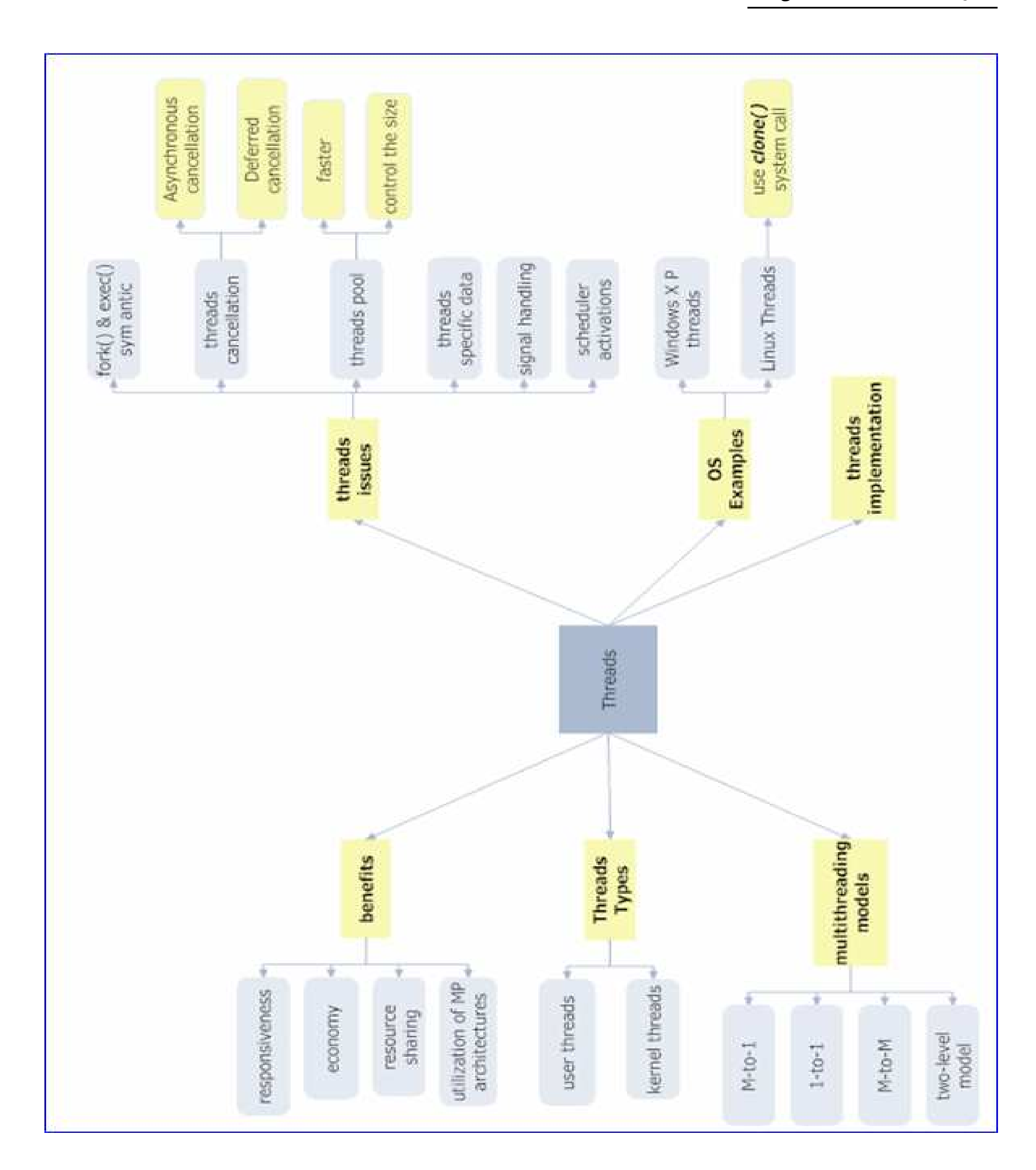
10- thread pool.

11- LWP.

12- مبدل(محول) السياق .

تم جمع الفصل بواسطة فاطمة الفرج (f.a.t.o.m.y@windowslive.com) و ابتهال نواف العتيبي (yasamenah@hotmail.com) وريم الرشيدي.

الخريطة الذهنية للفصل[31] :



[31] رسم قطر الندى السماعيل

المهمة أو العملية[32] (Process):

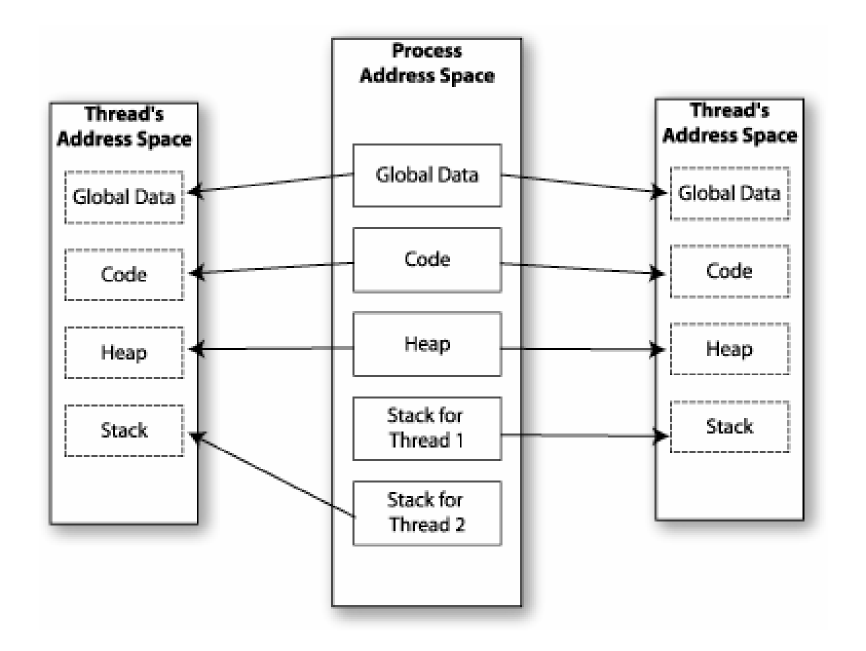
يتكون البرنامج (application) من واحد أو أكثر من المهام , ووجدت العديد من التعاريف للمهمة منها:

1-المهمة هو برنامج في قيد التنفيذ (an executing program) ويمتاز بأن له مصدر يأخذ منها البيانات مثل الذاكرة , أو أي مصادر يحتاجها في تنفيذ البرنامج.
2- (Process)هي عبارة عن بنية معطيات يقوم نظام التشغيل بإنشائها في الذاكرة الرئيسية ويوضع (mapping)التطبيق المراد تشغيله ضمن هذه البنية.
3- المهمة هو برنامج يعمل له تنفيذ على جهاز الكمبيوتر ، مثل انترنت إكسبلورر أو مايكروسوفت

الخيط أو سلسلة التعليمات[33](Thread):

وهي مجموعة من المهام التي ينقسم إليها البرنامج وذلك ليقوم بأكثر من مهمة بشكل متزامن حقيقي (وذلك عند وجود أكثر من معالج)أو تزامن كاذب (عند وجود معالج واحد)حيث يتم التبديل بين المهام بسرعة كبيره تعطينا انطباع بالتزامن.
وتعتبر (thread)الوحدة الأساسية لوحدة المعالجة المركزية.

[32]منى البريه
[33]لمياء السدحان



نلاحظ هنا بالرسم التوضيحي مهمة بأكثر من خيط ونلاحظ اشتراك الخيوط في الفضاء (address space) .

ولو ترجمنا thread نجد معناه خيط ! !

ماذا يعني خيط ؟......

هو المسار أو الطريق أو الخيط الذي يؤدي للمهمة نفسها , وهو يعتبر جزء من المهمة , وبما أنه مجرد طريق أو مسار أو خيط فبالطبع لا يحتاج مصادر مثل ذاكره حتى يأخذ البيانات منها , لأنه سوف يأخذ المصادر من مصادر المهمة نفسها .

ولكل خيط :

1- رقم (ID) للتعريف به

2- عداد(program counter)

3- سجل(register)

4- كومة(stack)

أما العلاقة بين الخيط والمهمة:

1- الـــخيوط تعتبر أجزاء من المهام بحيث أن كل جزء يقوم بمهمة معينه ، ولكن جميع الـــخيوط تستعين بالـــمهمة كمصدر لها , ويتم تحديد المهام من قِبل المبرمج أو البرنامج .

2 – العملية ستنفذ الخيوط (مجموعة من التعليمات) ، والذي يمكن أن يحتوي على عدة خيوط في بعض الأحيان.

3- جميع العمليات تتكون من واحد أو أكثر من الخيوط .

4- الخيوط والمهام كلاهما يشتركان في المعالج (SHARE CPU) وإنشاء طفل (creat child).

ما هو الفرق بين المهمة و الخيط[34]؟

1- الخيط هي أجزاء من المهمة

2- لكل مهمة عنوان(address space) مختلف و بيئة وقت التشغيل(runtime enivorment) ورقم تعريف (process ID) أما الخيوط طالما أنها مشتقة من المهمة لها عنوان واحد , ويشترك الخيط مع الخيوط الأخرى الموارد التابعة له.

3- لا يوجد خيط من غير مهمة لكن العكس ممكن

4- في حالة سياق التحول (context switch) مع المهمة يظهر over head , أما بحالة الخيط لا يظهر إلا إذا استدعينا نظام التشغيل وغالبا لا نستدعيه .

5- الخيوط هي أجزاء متزامنة التنفيذ داخل مهمة ما. (thread is a concurrent unit of process execution inside a)

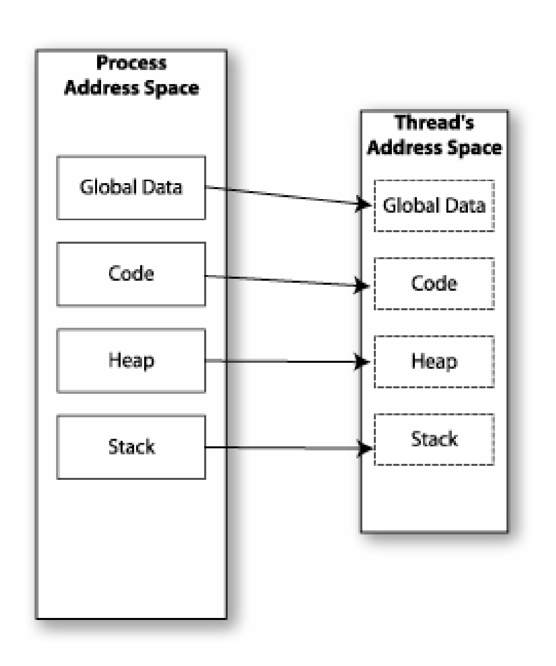
6- المهمة هي كلمة أعم و لفترة أطول , الخيط يقتصر على مفهوم "خط التنفيذ".

---

[34] منى البريه و بيان اليوسف

7- المهمة مرتبطة غالباً مع مستوى نظام التشغيل (مثل في متعدد المعالجة) ، بينما الخيط مرتبط بمستوى اللغة المنطقي المجرد.

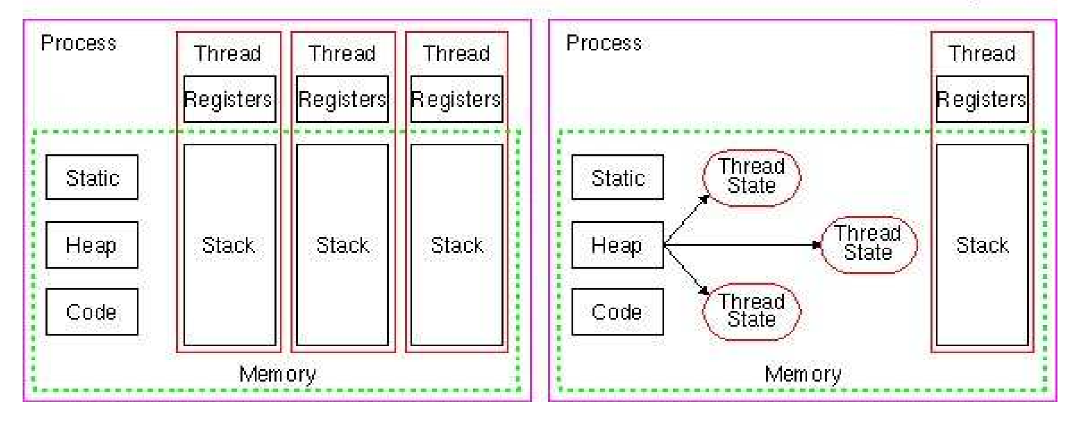
8- الخيوط ليست مستقلة مثل العملية.



نلاحظ هنا بالرسم التوضيحي مهمة واحدة وخيط واحد وهذا يعني اشتراك تام في كل شيء.



Model used in the Library



الصورة توضح أن الخيوط تشترك مع المهام في الفضاء ولكن هناك بعض العناصر تكون خاصة بخيط معين مثل:

سجل(register) و كومة (stack)

(تشترك سلاسل المهام التي في نفس العملية الـــمدونة(code) والبيانات(data) وموارد عملية التشغيل(files) .

وتستفرد عن بقية سلاسل المهام بـــالــ (register) و (stack) .)

(يوجد نوعين مختلفين للعمليات (processes)وهما[35]:

(1) عمليات مفرده المهام (single Threaded)

(2)عمليات متعددة سلاسل المهام(multi Threaded)

وتدعم العمليات المتعددة سلاسل المهام (multi Threaded)تنفيذ سلسلة مهام بشكل متوازي على عدة أنظمة حاسوبيه ويحدث عن طريق تعدد المهام (multitasking)أو ما يدعى بتجزئة الزمن (time slicing)

في الوقت الحالي تدعم العديد من أنظمة التشغيل تجزئة الزمن وتعدد المهام أو التنفيذ متعدد المعالجات(multiprocessor thread) عن طريق منسق العمليات (scheduler).

ومن الأمثلة التي توضح الإختلاف بين(multi Threaded) و(single Threaded):[36]

---

[35] بدور دهش الدهش و إيمان البلالي

[36] قطر الندى السماعيل

في الــ (single Thread) عندما نبحث في الويب (web server) فإنه لا يقبل إلا لعميل واحد فقط أن يستفيد في الوقت الواحد ولحل هذه المشكلة قمنا بإيجاد أكثر من سلسلة مهام (multi Threaded) في نفس العملية حتى تستجيب لطلبات العميل .

(من المعلوم أن (single threaded) تستطيع عمل فقط عملية واحده على (CPU) , بينما الــ (multi Threaded) تستطيع عمل العديد من العمليات على (multi-CPU) , ميزة (multi thread)أنها تستطيع زيادة بناء (multi process) .)

المميزات من إيجاد الــ (multi Threaded)[37]:

(1) رفع مستوى الاستجابة للمستخدم (responsiveness) :

إن إنشاء أفرع للمهمة النشطة يرفع مستوى الإستجابة للمستخدم حيث أن المستخدم أثناء تحميل صفحة من الإنترنت ولتكن على سبيل المثال صفحة موفري خدمة الرسائل الإلكترونية يمكنه أن يسجل الدخول لبريده الإلكتروني بينما الصفحة الرئيسية لموفر الخدمة لازالت تقوم بتحميل بعض الصور، كما يمكن للمستخدم أثناء استخدام أحد برامج معالجة النصوص أن يعطي أمر طباعة وفي نفس الوقت يمكنه طلب تدقيق إملائي ونحوي مما يساعد على رفع مستوى الإستجابة بشكل ملحوظ خلاف لو لم يكن هناك أفرع للمهمة النشطة لما تمكن المستخدم من إجراء أكثر من أمر أو تفاعل مع الجهاز في نفس اللحظة

(2) تقاسم الموارد (resourse sharing):

عند أنشاء أفرع للمهمة النشطة فإن هذه الأفرع تشترك افتراضيا في موارد النظام مثل الذاكرة مما يساعد على توفير هذه الموارد أي أنه يتم تقليل المساحة المحجوزة للمهمة الواحدة حيث أن جميع أفرعها تشترك في هذه المساحة مما يمكن المهمة النشطة من إجراء العديد من الوظائف في أفرعها المختلفة في نفس الحيز المحجوز ومن هنا يظهر تعددية الوظائف للمهمة الواحدة خلاف لو لم يكن هناك أفرع لاضطررنا لإنشاء

---

[37] أشواق العتيبي

أكثر من مهمة واحدة بعناوين متعددة في الذاكرة لأجراء تلك الوظائف مما يتسبب في إهدار موارد النظام , من المعلوم أن الــ (threads) يتقاسم الذاكرة والمصادر في نفس العملية والميزة من هذا التقاسم أنه يسمح للتطبيقات أن تملك العديد من سلاسل المهام النشطة في نفس العنوان(address space) .

(3) الاقتصاد (economy) :

من عملية إنشاء مهمة نشطة يستغرق وقت من النظام وذلك بسبب وجوب مراعاة حدود تلك المهمة حتى لا يتم اختراقها من قبل مهمة آخرى وكذلك عملية التبديل بين المهام المختلفة لرفع مستوى التز امنية في الأداء يضيف حملا أكبر على كاهل النظام . أما في إنشاء الأفرع لاتكون الإحترازات متشددة حيث أن جميع الأفرع تشترك في نفس مساحة الذاكرة إضافة إلى أن التبديل يكون أسرع بحوالي خمس مرات من التبديل بين المهام النشطة في بعض أنظمة التشغيل ويكون إنشاء المهام فيها أبطأ بثلاثين مرة من إنشاء الأفرع مما يجعل أنشاء أفرع متعددة للمهمة الواحدة لإجراء عدة وظائف مترابطة أرخص للنظام من إنشاء عدة مهام لإجراء تلك الوظائف .

(4)الاستغلال الأمثل في حالة وجود أكثر من وحدة معالجة مركزية في النظام :

تعدد المعالجات في النظام يرفع مستوى أداء الجهاز بشكل عام حيث أنه يمكن تنفيذ أكثر من مهمة نشطة في نفس الوقت مما يساعد على إنتاج العمل بشكل أسرع إلا أن المهمة الواحدة لم تستفد من هذه الميزة لأنه سيتم تنفيذها بشكل متتابع ولكن إذا قمنا بإنشاء أفرع للمهمة الواحدة سيتم تسريع تنفيذ المهمة نفسها حيث أن كل فرع من أفرعها تتم معالجته في معالج منفصل مما يتيح لها الاستفادة من الموارد على الوجه الأمثل .

مكتبة سلسلة المهام:

توفر مكتبة سلسلة المهام للمبرمج واجهة برمجة تطبيقات لخلق وإدارة سلسلة المهام.

– هناك طريقتين أساسيتين لتطبيق سلسلة المهام هما:

1- توفير المكتبة كاملة في مساحه المستخدم بدون أي دعم من النواة:
كل الرموز وهياكل البيانات الخاصة بالمكتبة موجودة في مساحة المستخدم.
2- إنشاء مكتبة على مستوى النواة بدعم مباشر من نظام التشغيل:
كل الرموز وهياكل البيانات الخاصة بالمكتبة موجودة في مساحة النواة.

يوجد ثلاث مكتبات تستخدم حالياً هي[38]:

1- POSIX Pthread
توفر إما مكتبه على مستوى المستخدم أو على مستوى النواة
2- WIN32
مكتبه على مستوى النواة وهي متاحة في نظام النوافذ.
3 - Java. Pthread
واجهة برمجة التطبيقات لسلسله مهام الجافا تتيح إنشاء وإدارة سلسلة المهام مباشره في برامج الجافا.

وهناك طريقتان لإنشاء تجزيء للعمليات[39]:

1- (User threads) تجزيء المستخدمين :
وهي عملية تجزيء للبرامج من خلال المستخدمين داخل برنامج معين دون المرور بلب النظام (Kernel) بواسطة مناداة الدوال المكتبية
(library function) وتعتمد هذه الطريقة علــى نوع نظام التشغيل المستخدم ولا تتحكم في أجزاء النظام المادية (hard ware).
وفائدة هذا النوع انه يخفف من الضغط على لب النظام (Kernel)

2- (Kernel threads) تجزيء النظام :

[38] أمل الحماد
[39] فاطمة الفرج

وهي عملية تجزيء للبرامج من خلال لب النظام (Kernel) بواسطة مناداة الدوال (function) المسؤولية عن إنشاء تجزيء للعمليات وتسمي هذه الطريقة (system call)

ومن الأمثلة عليها :

- Windows XP/2000, Solaris, Linux, Mac OS X
- وكلتا الطريقتين تنفذان من خلال لب النظام (Kernel)

(هناك نوعان للــ(thread)سلسلة المهام[40]:

- سلسلة مهام مجال المستخدم (user_space thread) : تُنتج في مستوى المستخدم .
- سلسلة مهام النواة (kernal) : معموله من قبل النواة (kernal)

ويتم الربط بين سلسلة مهام مجال المستخدم (user_space thread) و سلسلة مهام النواة (kernel) عن طريق ثلاث طرق .

نماذج الخيوط المتعددة (multi Threading Models)[41]:

1) متعدد إلى واحد Many-to-one Mode :



في هـــذه الطريقة يوجـــــد العديد من (user threads) تنتـــقل إلى (kernel threads) , وهذه الــ (thread) تدار في مساحة المستخدم عن طريق (library) , لكن هذه الطريقة الواحد يعني لا يمكن أن يكون تنفيذ العديد من العمليات بشكل متوازي.

---

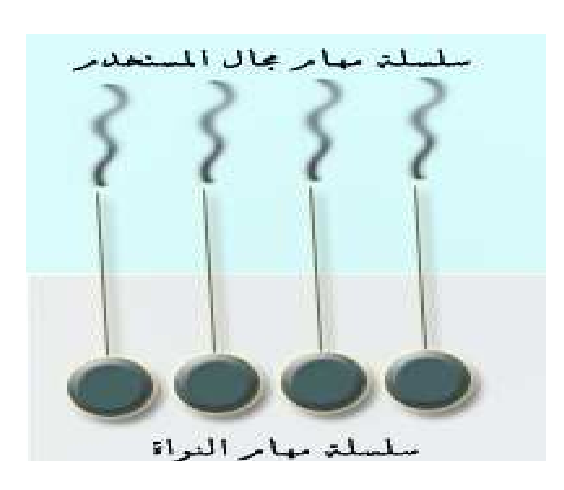
[40] منى الماجد
[41] غادة القرعاوي

عيوبها :فيها إغلاق بسبب أنه فقط واحد من سلسلة المهام يستطيع المرور إلى (kernel)في الوقت ذاته

وهذه الطريقة تستخدم في:

.Solaris Green Threads, GNU Pthreads

<u>(2)One-to-one Mode   من واحد إلى واحد</u>



في هذه الطريقة كل (user threads) يملك (kernel threads) له لوحده ,

وبهذه الطريقة يمكن لكـــــل سلسلة مهام أن تعمل بشكل متوازي مع الأخرى,ولذلك لابد

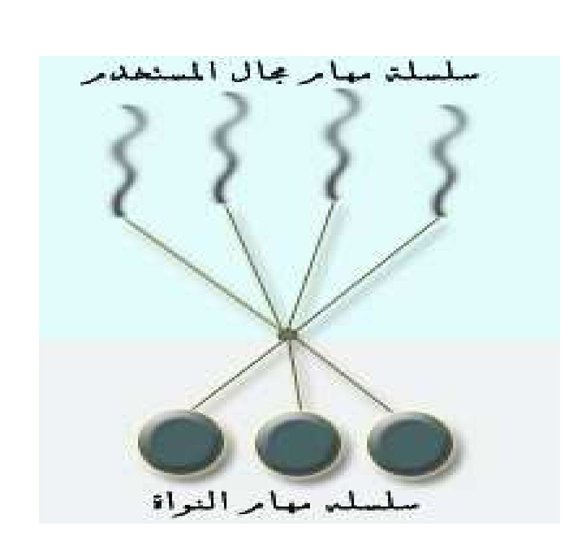
مع كل (user threads) جديد أن نوجد (kernel) متوافق معه , هذه الطريقة أفضل من السابقة .

مشكلة هذه الطريقة:

أنه في حالة تعطل الـــ (kernel) لا يمكـــن لـــ (user thread) التابعة له أن تستفيد من أي (kernel) أخر و تسبب ضغط على لب النظام (Kernel).

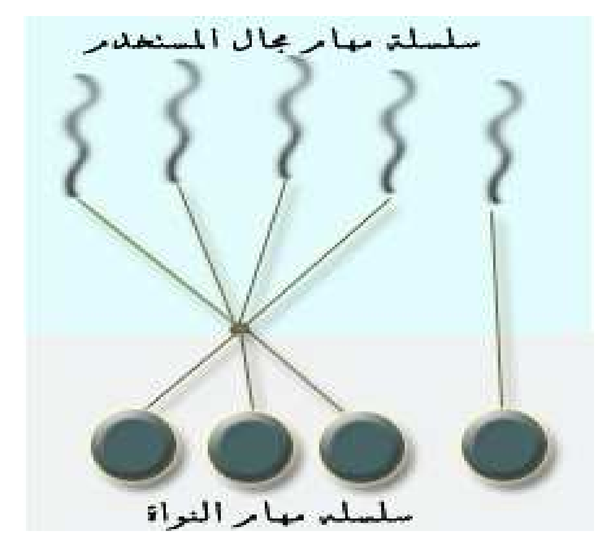
وتستخدم هذه الطريقة في: Linux, windows 95,98,NT,2000,XP .

(3) من متعدد إلى متعدد Many-to-many Model :



في هذه الطريقة يوجد العديد من (user threads) التي تملك نـفـــس عددها أو أقـــل منهـــا (kernel threads) وهـــذه الطريقة أفضـــل مـــن السابقتين لأنه كــل (user threads) ممكن يتعامل مع أكثر من (kernel).

(4) نموذج الطبقتين (Two-level Model) (Similar to M:M) مشابه لـ M:M



كما أنه في بعض الأحيان يجمع بين طريقتين وهذا النوع يعطي صلاحية لأن ينعزل كل جزء من أجزاء البرنامج ويتصل بلب نظام خاص به.

وهذا مدعوم في أنظمة التشغيل:-IRIX,HP-UNIX UX and Tru64 .)

تحتاج (M: M) و(Two-level-model) إلى وسيلة اتصال أو ربط , لكي تحدد العدد المناسب من الــ kernel thread التي تريدها لعمل نشاطاتها ,
لذلك تستخدم LWP وهي عبارة عن تراكيب بيانات يقوم الــ kernel بخلقها

وذلك لربط الـــ kernel thread بالـــ user thread .

الفرق بين خيوط قلب النظام وخيوط المستخدم[42]

User Level Threads vs. Kernel Level Threads

	User level thread	Kernel level thread
الإدارة	يتم بواسطة مكتبة الـ User level thread	تتم بواسطة نظام التشغيل
المساحة	مساحة المستخدم	مساحة نظام التشغيل
تدخّل نظام التشغيل	لا يوجد تدخّل من نظام التشغيل	مباشرة مدعومة من قِبل نظام التشغيل
الميزات	سرعة الإنشاء والتحكم	إذا كان thread يؤدي إلى منع مناداة النظام فإن قلب النظام (النواةKernel) يمكن أن يحدد thread آخر في التطبيق للتنفيذ.
المساوئ	إذا كان قلب النظام وحيد(single threaded)، فإن أي user thread أدى لمنع مناداة النظام سيتسبب في منع كل العملية، وحتى لو وجد other threads متاحة للتشغيل في إطار التطبيق.	أبطأ في الإنشاء والتحكم

[42]حنان وحيد الهندي

أمرا التفرع والتنفيذ[43]:

fork() and exec()

عند إنشاء مهمة نشطة بأمر التفرع فأنه يتم نسخ جميع محتوى المهمة الأم إلى المهمة المنشاة
التساؤل المطروح هنا : هل سيتم نسخ جميع الأفرع لهذه المهمة إلى المهمة المنشأة أم سيتم فقط نسخ الفرع الذي أصدر الأمر بإنشاء مهمة جديدة ؟

إجابة التساؤل تعتمد على التطبيق المتضمن لهذه المهام , إن تم إصدار أمر التنفيذ مباشرة بعد أمر التفرع فإن محتوى المهمة المنشأة سيتحول بكامله إلى البرنامج المرسل له في أمر التنفيذ مما يعني أنه لا أهمية لأفرع المهمة الأم لهذه المهمة المنشأة ففي هذه الحالة سيتم فقط نسخ الفرع الذي أصدر أمر التفرع في المهمة الأم إلى المهمة المنشأة , أما إذا لم يتم مناداة أمر التنفيذ بعد أمر التفرع مباشرة فإنه يتوجب نسخ جميع أفرع المهمة الأم إلى المهمة المنشأة .

(Cancellation) الإلغاء:

عندما تكون هناك رغبة في إنهاء التفرع على غير المناخ الطبيعي وهو إتمام وظائفها فإن هذا الإنهاء يكون إلغاء في بعض الأحيان يرغب المبرمج في إلغاء التفرع لعدم أهميته كأن يكون أنشأ أكثر من فرع للبحث في قاعدة بيانات , فعندما يعود أحد هذه الأفرع بالنتيجة فإنه لا حاجة لبقية الأفرع فيلجأ المبرمج إلى إلغائها مباشرة لتحرير موارد النظام المحجوزة لها , إلا أنه بهذه الطريقة لن يتم تحرير جميع هذه الموارد علاوة على أنه قد يكون هذا الفرع المراد إلغائه "الهدف" يعمل على تعديل قيمة مستخدمة من قبل الأفرع الأخرى مما سيؤدي إلى أخطاء لا حصر لها , لذلك فُضل أن يكون الإلغاء مؤجل مما يعني أن يعطى المجال للفرع الهدف بأن ينهي نفسه في اللحظات الأكثر أماناً وذلك يكون بإعطاء علم يدل على إمكانية إنهائه في هذه اللحظة أو الانتظار قليلاً ريثما يكون الوضع أكثر أمناً .

[43] حنان وحيد الهندي

ويوجد هنـــــاك حالتين مختلفتين عند الإلغـــــاء[44]:

1-(Asynchronous cancellation) :
وهذي تعني أن سلسلة المهام تلغى مباشرة بمجرد طلب إلغـــــائـــــها.
2-(Deferred cancellation):
وهذي تعني أن سلسلة المهام لا تلغى مباشرة إلى أن تنتهي من عملها الذي تقوم به لأنه عند إنهائهـــــا قد تؤدي إلي تضرر الملفات.

مثال على طريقة كتابته:

```
#include <pthread.h>
int pthread_cancel(pthread_t thread);
```

لإيقاف عملية في نظام لينكس يوجد عدة طرق[45]:

1- CTRL-C (for foreground processes)
هذا الأمر لإيقاف العملية الحالية – التي في الواجهة

2-(Kill process-identifier (for Background processes
وهذه ابسط وأسهل وأنظف طريقة لإيقاف أو قتل عملية

وإذا أردتي معرفة الـــ Process-identifier
باستخدام الأمر ps
سوف يظهر لنا جميع العلميات التي تعمل لدينا في النظام و أول عمود هو الذي به الرقم الخاص لكل عملية process-identifier

وإذا لم ينجح هذا الأمر استخدمي
Kill -1 process-identifier

[44] إيمان البلالي
[45] بتول الحناكي

هذا الأمر سوف يخبر العملية بأن تستسلم لأنها سوف تعطيها إشارة بأنك قد خرجت من النظام فلابد لها من الاستسلام بدلا من أن تحاول إكمال عملها وهذا الأمر سوف يقتل كل الـــ child التابعين لهذه العملية

Kill -9 process-identifier

وهذا الأمر سوف يوقف العميلة ولكنه لن يوقف عمل أي child تابعين لها

وتستطيعين أن توقفي عمليتين في نفس الوقت

Kill process-identifier process-identifier

((Signal handling) حامل الإشارة[46]:

الإشـــــارة تستخدم في نظام اليونكس لإعطاء ملاحظه لـــــ العملية في حالة حدوث شيء ما مثل الإلغاء , الانقطاع وقد تستقبل الإشارة إما متزامنة أو غير متزامنة.

النوع الأول : (التزامن) الإشارة تكون بداخل العملية .

النوع الثاني: (غير متزامن) الإِشـــــــارة تأتي من خارج العملية.

<u>كل إشارة قد تُحمل بواسطة أحد الحاملين:</u>

1- (default): قد يعالج حاله واحدة أو حالتين فقط.

2-(user-defined): قـــــــــــــد يعالج حـــالات كثيـــــــــــرة.

<u>أمثلة على الـــــــ Synchronous Signals:</u>

هذا النوع من الإشارات يظهر عند الدخول غير المسموح به لجزء من الذاكرة أو عند القسمة على الصفر مثلا.

[46] رويدا العندس

وبهذا إذا قام برنامج معين بحدث من هذا النوع, لابد أن تظهر إشارة , وفي هذا النوع من الإشارات , توصل الإشارة لنفس العملية التي تسببت بظهورها ولهذا السبب سميت بالـــــ Synchronous Signals.

أمثلة على الــ Asynchronous Signals:

في هذا النوع, السبب الذي يؤدي لظهور الإشارة يكون (خارج العملية) .
مثلا,عند الضغط على أوامر معينة مثل Control + c وهذا يؤدي إلى إنهاء عملية ما بشكل مفاجئ.

تـــــأثيـــــر الإشـــــــارة له أربع حالات:

1- الإشارة ترسل فقط لكل سلسلة المهام التي حدث فيها الخطأ فقط.
2- الإشارة ترسل لكل سلسلة مهام موجودة في العملية.
3- الإشارة ترسل لسلسة مهام معينه في العملية.
4- الإشارة ترسل لسلسلة مهام مخصصه فقط لاستقبال الإشارات.

على سبيل المثال, الإشارات من نوع Synchronous Signals لابد أن ترسل إلى الجزء الذي سبب ظهـــــــور الإشارة وليس لباقي أجزاء العملية.
أغلب أنظمة اليونكس تسمح للــــ thread بـــــأن يقوم باستقبال إشارات معينة ويحجب إشارات أخرى. ولهـــــذا في بعـــض الأحيـــــان يتـــم إرســـــال Asynchronous Signal فقط للأجزاء التي لم تحجبها . وبما انه لابد من معالجة الإشارة فإنه يتم تسليمها لأول thread لم يقم بعملية الحجب.
في أنظمة الويندوز, يتم معالجة الإشارات عن طريق الـــــ APCs (Asynchronous procedure calls).
حيث تقوم هذه الخاصية بالسماح للـــــ user thread بتحديد (function) ليتم مناداتها عندما يستلم جزء معين التنبيه لظهور حدث.ويتم التسليم لجزء معين بدلا من التسليم لكل العملية.

مثال على استخدام الإشارة[47]:

عندما يحدث للجهاز حالة تعليق فإن ctrl+z أوctrl+c ستوقف التعليق وتعيد العمل للجهاز مرة أخرى.

أما عند عملي في الخلفية فإن ctrl+z لن تنفع لذلك سوف أكتب kill

وإذا كنت أعمل في الــ pash فيجب أن اعرف رقم الــ pash حتى أعمله kill وأكتب : kill ثم رقم الpash -->(kill 9) .)

:Thread pool

في السابق قبل عمل ال (pool) كان كل عميل جديد ننشأ له سلسلة مهام جديدة مثل عندما يكون لدينا مائه عميل حيث ننشأ له مائه من سلسلة المهام وهذا العمليه كانت لها العديد من العيوب .

وهــــــــي:

1- تستهلك وقت لإنشاء سلسلة المهام (cpu time) .

2 - تأخذ مســـــاحه من العملية .

لذلك تم إيجاد (pool) وهي إيجاد عدد محدد من سلسلة المهام عندما نبدأ العملية وننقلها إلى (pool) وعندما يتم طلب خدمه فإن سلسلة المهام تأتي من الــ (pool)إذا كانت ممكنه وتمر لخدمة الطلب , وعندما تكمل عملها تعود إلى الــ (pool) لتنتظر طلب جديد .

محاسن استخدام الــ(pool) :

1- أن طلب الخدمة من سلسلة مهام موجودة أسرع من انتظــــار إنشاء الخدمة سلسلة مهام جديدة.

2-يوجد في الــ(POOL) عدد محدد من سلسلة المهام منشأه في نقطة ما .. أفضل من إنشاء عدد لا نهائي من السلسلة المهام.

---

[47] منى حكمي , حليمه حكمي ، قطر الندى السماعيل

طرق تحديد عدد الأجزاء المسموح بها:

يتحكم بها عدة أشياء منها عدد الـــــــ(CPUs) المتوفر في النظام , حجم الذاكرة وعدد الطلبات المتوقعة من المستخدم.
وهناك أنظمة ذكية والتي تقوم بدورها بتحديد العدد المطلوب تبعا للاستخدام وهذا ينتج (pool) أصغر وبالطبع يقوم بتوفير جزء من الذاكرة المستخدمة عندما يكون الضغط قليل على النظام.

بالنسبة للبرامج المتعددة الأجزاء هنا مشكلة التواصل بين قلب النظام والـــــــ(thread library) والتي تتطلب من النظام بالقيام بتحديد عدد من أجزاء قلب النظام للقيام بعملية الارتباط.

هناك عدد من أنظمة التشغيل تقوم بوضع بيانات تتوسط بين أجزاء المستخدم و أجزاء قلب النظام. هذه البيانات والتي تعرف بالــــ Lightweight processأو LWP عبارة عن معالج خيالي والذي يقوم البرنامج الذي يكون تحت التشغيل بجدولة جزء المستخدم عليه .
وبهذا كل LWP تكون مرتبطة مع جزء من قلب النظام.

قد يتطلب برنامج معين اي عدد من الــــLWP لكي يتم تشغيله بطريقة صحيحة.

على سبيل المثال:

في حالة وجود برنامج يعمل على معالج واحد , أذن لن يتم تشغيل أكثر من جزء في المرة الواحدة ولن يحتاج إلى أكثر من LWP واحد.

Scheduler Activation[48]:

طريقة للتواصل بين الـUser thread library وقلب النظام

طريقة عمله:

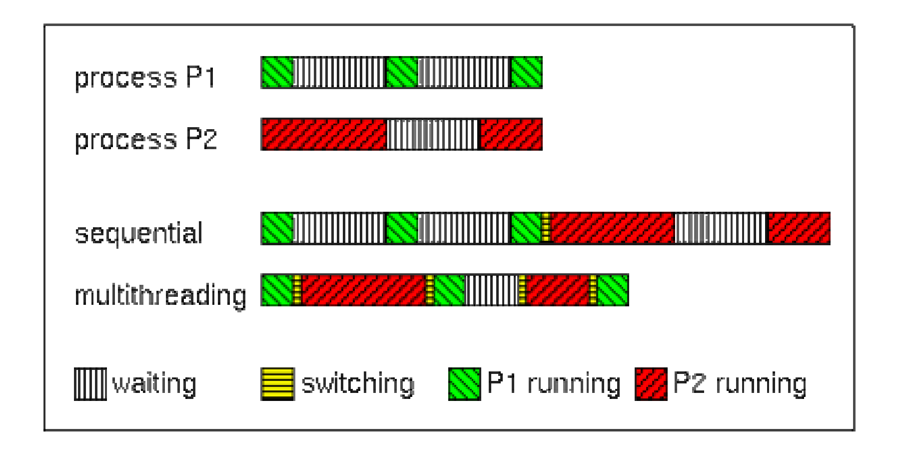
يقوم قلب النظام بتزويد البرنامج بمجموعة من الــــLWP وعندها يستطيع البرنامج بالقيام بجدولة أجزاء المستخدم على المعالج الخيالي .

Upcall: هي الحالة المعروفة عندما يقوم قلب النظام بإعلام برنامج معين بظهور حدث جديد.

طريقة عمل هذه الخاصية:

تتم معالجة التنبيهات القادمة من قلب النظام من قبل **Upcall handler** والتي لابد من أن تعمل على المعالج الخيالي.

مثال على حدوث عملية Upcall: عندما يقوم قلب النظام بالتنبيه أن هناك جزء معين سوف يتم إيقافه , في هذه الحالة سيتم تعيين معالج خيالي أخر لهذا البرنامج.



[48] رويدا العندس

:context switch and multi-threading[49]

عند استخدام multi-threading فان TCB يعتبر جزء من Context switch , صف الانتظار والصف الجاهز المستعد يحويان مؤشرات تشير إلي TCB مبدل السياق وهو المسؤول عن عمل نسخة لحالة وحدة المعالجة المركزية من وإلى TCB .

كيفية عمل مبدل السياق:

مبدل السياق بين تشعبين ينتمون لعملية واحدة : في هذه الحالة لا نحتاج لتغيير فضاء العناوين .

مبدل السياق بين تشعبين ينتمون لعمليتين مختلفتين : سوف نحتاج لتغيير فضاء العناوين .

بعض أسباب استخدام المهام المتعددة (threads) في تصميم نظام التشغيل[50]:

1- تقاسم الموارد : حيث أن مهام البرنامج الواحد تتشارك في أجزاء عده كالذاكرة .

2- توفير الوقت : حيث يتم عمل عدد من المهام في نفس الوقت .

مثل : عندما نكتب ببرنامج الـــword في وقت تشغيله يصبح هو process وعندما نكتب فيه تكون هناك عمليات تعمل بنفس وقت عمليه الكتابة

مثل تنفيذ save every two min & spell check هنا عمل شيئين in one process.

3-العملية(process) مع المهام المتعددة تجعل الخادم(server) يقوم بعمله بفاعلية أكبر.

4-.بما أن المهام المتعددة تتشارك في البيانات(data) فإنها لا تحتاج (interprocess communication).

5- عند إيقافه أو إنهاءه يأخذ وقت أقل من processes .

6- لا حاجة للاستعانة بـــ kernal أو إخباره بأي عملية تتم عن طريق thread لأنه لا يعلم بوجودها .

---

[49] منى الماجد

[50] بدور دهش الدهش , إيمان البلالي ، أفنان البحيري

المهام المتعددة رخيصة وذلك لأن[51]:

1- تقلل التكاليف من حجز الذاكرة حيث تكون مشتركه لعدد من المهام.

2- نحتاج فقط إلى(stack) و أماكن تخزينية في (register) لذا فإن المهام المتعدده رخيصة الإنشاء.

3- المهام المتعددة تستخدم جزء بسيط من مصادر نظام التشغيل عند عملها حيث لا تحتاج إلى(address space) جديد ولا بيانات عامة (data global) ولا (program code).

4- التبديل سريع عندما نعمل مع المهام المتعددة وذلك لأننا نحتاج فقط إلى تخزين أو إعادة تخزين pc (program counter) , sp (stack pointer) , register.

---

[51] إيمان البلالي

الفصل الخامس:
جدولة وحدة المعالجة المركزية

جدولة وحدة المعالجة المركزية

CPU Scheduling

مقدمة:

وحدة المعالجة المركزية (CPU) من أهم أجزاء الحاسوب و التي تقوم بتنفيذ العمليات (processes) و لكن وحدة المعالجة المركزية (CPU) تقوم بتنفيذ عملية (process) واحدة في الوقت الواحد و هناك الكثير من العمليات التي تحتاج للتنفيذ، فكيف يتم التنسيق و التنظيم بين العمليات؟!

إن المجدول (CPU scheduler) هو المسؤل عن التنسيق بين العمليات باستخدام خوارزميات مختلفة، في هذا الفصل سنتناول بعض المفاهيم الأساسية في وحدة المعالجة المركزية، و أهم خوارزميات الجدولة و ميزاتها و عيوبها و الفرق فيما بينها.

أهداف الفصل:

- التعرف على مفاهيم مهمة في وحدة المعالجة المركزية
- التعرف على جدولة وحدة المعالجة المركزية (CPU scheduling) و التي تعتبر أساس في أنظمة التشغيل ذات البرامج المتعددة (multiprogramming)
- التعرف على خوارزميات جدولة وحدة المعالجة المركزية المختلفة
- معرفة كيف يمكن اختيار الخوارزمية بناءً على النظام المتاح لدينا

محتويات الفصل:

- مفاهيم أساسية
  - حلقة العمل داخل وحدة المعالجة المركزية – وعملية الإدخال والإخراج (CPU-I/O Burst Cycle)

- جدولة وحدة المعالجة المركزية (CPU Scheduler)
- الجدولة الإجهاضية (Preemptive Scheduling)
- ما هو الdispatcher؟

▪ معايير الجدولة (Scheduling Criteria)

▪ خوارزميات الجدولة (Scheduling Algorithms)

- من يأتي أولاً يخدم أولاً ((First-come First-served (FCFS-FIFO))
- العمليات ذات الوقت القصير أولاً ((Shortest-Job-First (SJR))
- راوند روبن (Round Robin (RR))
- طابور متعدد المستويات (Multilevel Queue)
- طابور متعدد المستويات مع إمكانية التنقل(Multilevel Feedback Queue)

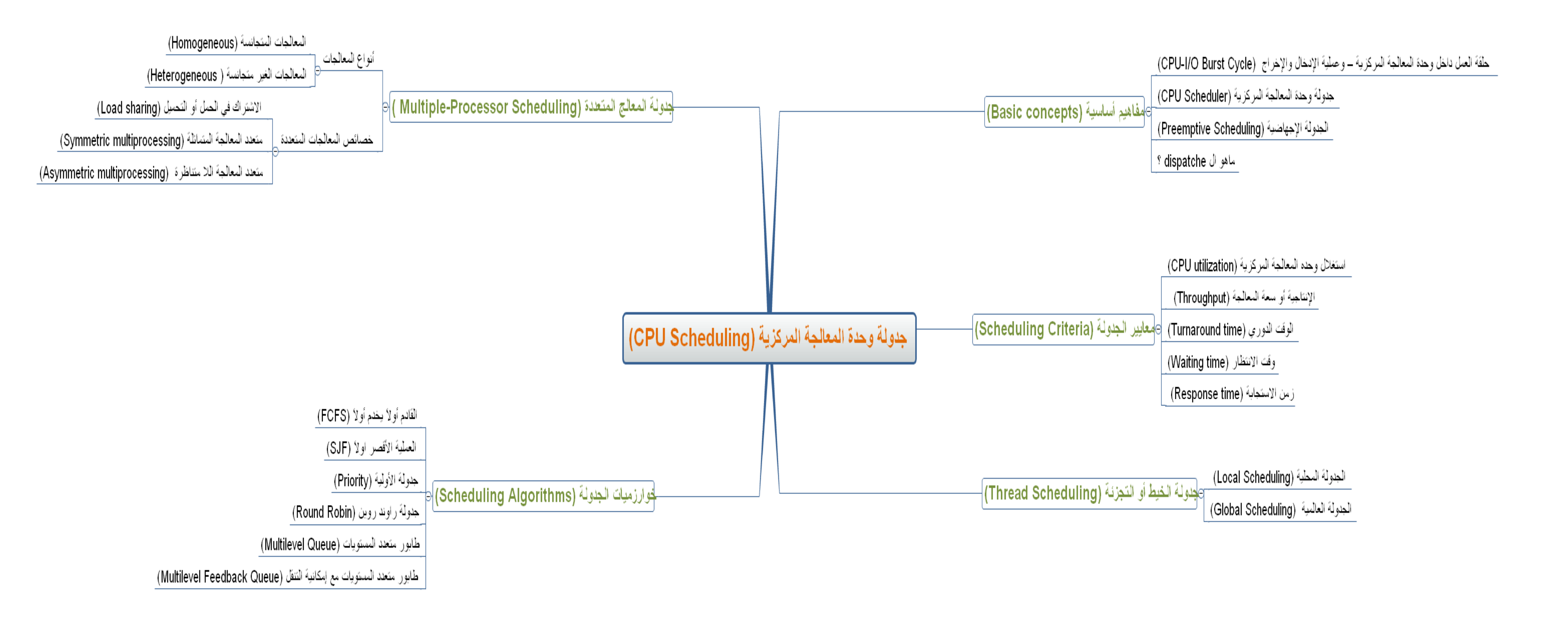
▪ جدولة المعالج المتعددة (Multiple-Processor Scheduling)

▪ جدولة الخيط أو التجزئة (Thread Scheduling)

تم تجميع هذا الفصل بواسطة خلود صالح الرومي (k.alroumi@gmail.com)

((اللهم إن أخطأت فمن نفسي والشيطان فاغفر لي وتب علي وإن أصبت فذلك فضل منك تفضلت به علي فلا تحرمني الأجر وزيادة الفضل))

الخريطة الذهنية لجدولة وحدة المعالجة المركزية (CPU Scheduling Mind Map)



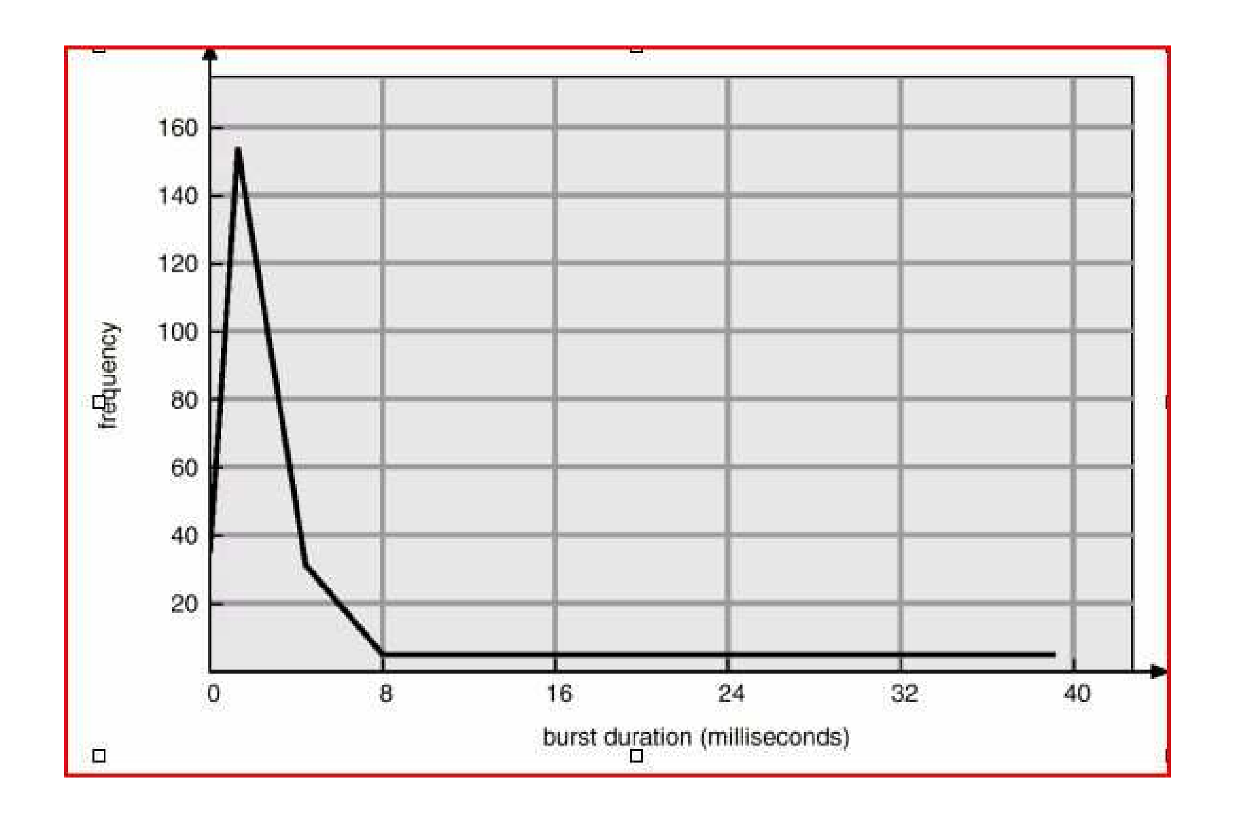
حلقة العمل داخل وحدة المعالجة المركزية – وعملية الإدخال والإخراج (CPU-I/O Burst Cycle):

تنفيذ العملية(Process) يتألف من حلقة من معالجة العملية داخل وحـــدة المعالجـــة المركزيـــة (CPU) وانتظار عملية الإدخال و الإخراج.

كل عملية تمر بدورة متبادلة مابين العمل داخل وحدة المعالجة المركزية (CPU burst) و العمل في وحدات الإدخال و الإخراج (I/O burst) ليتم تنفيذها ، فعند طلب فتح البرنـــامج تبـــدأ العملية بالعمل داخل وحدة المعالجة المركزية (CPU burst) ثم العمل في وحدات الإدخال و الإخراج(I/O burst) ،ويستمر في الحالتين بشكل متبادل ، وينتهي بالعمل داخل وحدة المعالجة المركزية (CPU burst) بطلب نظام التشغيل (OS) لإنهاء عملية التنفيذ لسبب ما، إما لأن العملية انتهت أو لحدوث خطأ تسبب بإلغائها .

إذا كانت العملية معظم وقتها تقضيه في عمليات الإدخال والإخـــراج (I/O bound) ، فإنهـــا تستغرق مدة قصيرة في العمل داخل وحدة المعالجة المركزية (CPU burst) ، و وقت أطول في العمل في وحدات الإدخال و الإخراج (I/O burst) ، وإذا كانت العملية معظم وقتها تقـــضيه في وحدة المعالجة المركزية (CPU bound) ، فسيحدث العكس .

.بمعنى آخر ، عند تنفيذ عمليات معظم وقتها تقضيه في وحدة المعالجة المركزية (CPU bound) فإنها تقضي وقت طويل في العمل داخل وحدة المعالجة المركزية (CPU burst) ولكـــن عـــدد مرات تنفيذ العمل داخل وحدة المعالجة المركزية (CPU burst) قليل ، والعمليات التي معظـــم وقتها تقضيه في عمليات الإدخال والإخراج (I/O bound) تقضي وقت قصير في العمل داخل وحدة المعالجة المركزية (CPU burst) ولكن عدد مرات تنفيذها (CPU burst) أكثر.



يبين المنحنى أن البرامج التي تكون مده عملها داخل وحده المعالجة المركزية (CPU burst) طويل نادرة الاستخدام، بينما البرامج التي تكون مده عملها داخل وحده المعالجة المركزية (CPU burst) قصير تكون كثير الاستخدام، و(frequency) يوضح تكرار ظهور برنامج معين.

يمكننا الآن تصنيف العمليات إلى:

1- عمليات معظم وقتها تقضيه في وحدات الإدخال و الإخراج (I/O bound processes) :

تستخدم وقت أكثر لإتمام عملات الإدخال والإخراج, هذا النوع من العمليات يشغل وحدة المعالجة المركزية (CPU) قليلا لذلك فهي تملك عمليات قصيرة و كثيرة من ال CPU bursts

2- عمليات معظم وقتها تقضيه في وحدة المعالجة المركزية (CPU bound processes):

تستخدم وقت أكثر لإتمام العمليات الحسابية، تشغل وحدة المعالجة المركزية (CPU) كثيرا لذلك فهي تملك عمليات طويلة و قليلة من ال CPU bursts

بالإضافة إلى التصنيفين السابقين يوجد تصنيفات أخرى بديلة:

1- العمليات التفاعلية (interactive processes):

تتفاعل هذه العمليات بشكل ثابت مع المستخدم، لذلك تستهلك وقت أكثر في انتظار حدث ما من لوحة المفاتيح أو الفأرة، عندما تستقبل العملية المدخل(input) يجب أن تــستيقظ (wake up) بسرعة بمعدل تأجيل(avg delay) بين 50-150ms وإلا فإن المستخدم سوف يعتبر النظام غير مجدي.

البرامج التفاعلية غالباً ما تكون برامج تحرير النصوص،برامج تخطيطية.

2- عمليات الدّفعات أو المجموعات (batch processes):

لا تحتاج تفاعل مع المستخدم، بالتالي يتم تنفيذها في الخلفية، يتضح من السابق أن هذه العمليـــات لا تحتاج استجابة سريعة، لذلك يتم تأجيلها من قبل المجدول.

البرامج التي تستخدم عمليات الدّفعات أو المجموعات (batch processes) غالباً ما تكون لغات البرمجة، محرك بحث قواعد البيانات.

3- عمليات الوقت الحقيقي (real-time processes):

تحتاج إلى متطلبات جدولة قوية جداً ، مثل هذه العمليات لا يتم إعاقتها من قبل عمليات ذات أولوية أقل ويتضح مما سبق أن الاستجابة يجب أن تكون سريعة.

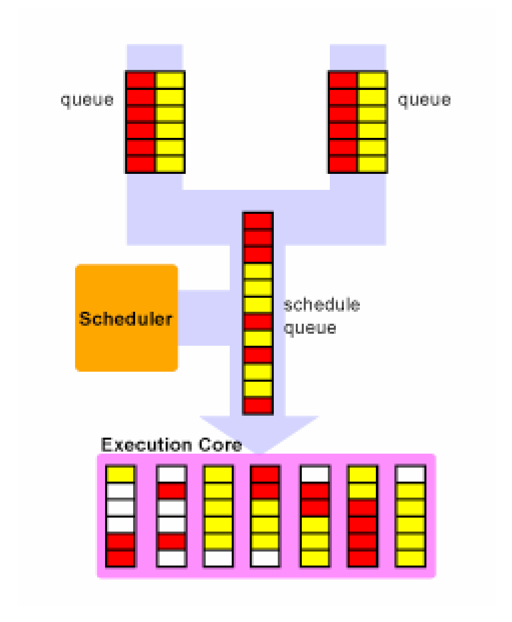
سميه الخنيزان- أروى السلامة - هند المطيري – منى الماجد

المراجع:

Operating System Concepts

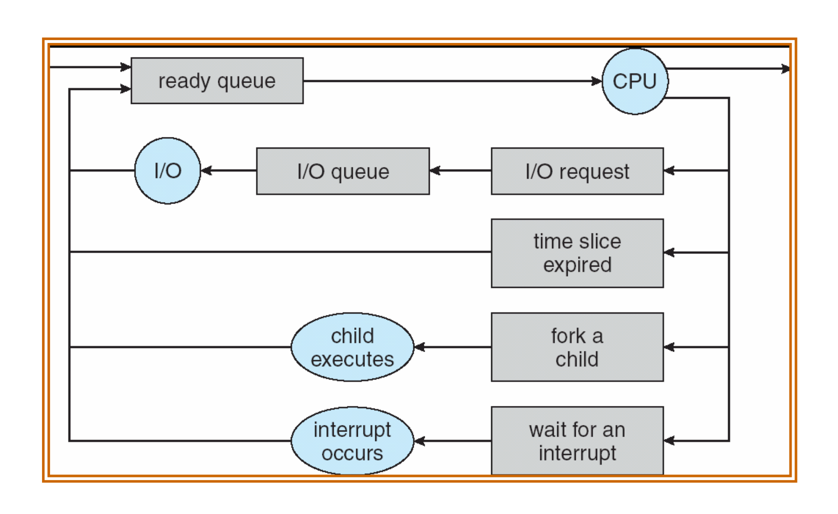
جدولة وحدة المعالجة المركزية (CPU Scheduler)

هي وضع خطة لترتيب دخول العمليات على المعالج بحيث تدخل عملية واحده كل مرة و نـــستغل معظم وقت المعالج ، و يقوم مجدول العمليات بترتيب دخول العمليات على المعالج



جدولة الطوابير:

عندما تدخل عملية إلى النظام فإنها تدخل في طابور المهام الذي يحتوي جميع عمليات النظام ،و عندما تصبح العملية جاهزة و تنتظر التنفيذ فإنها تنتقل إلى الطابور الجاهز، أما إذا كانت العملية تنتظر عملية إدخال أو إخراج مثل التحميل من القرص الصلب أو كانت تخدم اتصال انترنت فإنها تنتقل إلى طابور الأدوات



عندما تصبح وحدة المعالجة المركزية (CPU) عاطلة (idle) فإن نظام التشغيل يختار عملية من العمليات الموجودة في الطابور الجاهزة للتنفيذ (ready queue) .

بواسطة خاصية المجدول القصير الأمد(short-term scheduler) يتم اختيار عملية من العمليات الموجودة في الذاكرة وتحديداً في الطابور الجاهز(ready queue) ووضعها في وحدة المعالجة المركزية (CPU) لتنفيذها.

أنواع الجدولة:

1- جدولة طويلة الأجل(long-term scheduler):

وظيفته تحميل العمليات (processes) من الذاكرة، و هي التي تقرر أي العمليات ستدخل إلى الطابور الجاهز و أيها تخرج أو تتأخر، و هذه الجدولة ليست موجودة في الحاسبات المكتبية فالعمليات تدخل إلى المعالج آلياً و هي مهمة لنظام الوقت الحقيقي و الالتزام بمواعيد العمليات النهائية.

2- جدولة قصيرة الأجل(short-term scheduler) :

وظيفته يختار العملية (process) الجاهزة لكي تشتغل على المعالج CPU (هذا النوع سريع جدا).

تقرر أي العمليات الجاهزة سيتم معالجتها بعد إشارة المقاطعة أو بعد استدعاء النظام، و هي أسرع من الجدولة الطويلة أو المتوسطة حيث تأخذ القرارات في وقت قصير جداً ، ويمكن أن تكون قادرة على إجبار العمليات على الخروج من المعالج و إدخال عمليات أخرى أو تسمح ببقاء العمليات في المعالج حتى تنتهي.

3- جدولة متوسطة الأجل (medium-term scheduler):

هذه الجدولة موجودة في كل الأنظمة ذات الذاكرة الافتراضية ، فهو يقوم بعملية التبديل أي أن يزيل العمليات بشكل مؤقت من الذاكرة الرئيسية إلى الذاكرة الثانوية و ذلك حسب أولوية العملية و ما تحتاجه من مساحة على الذاكرة.

في هذه الأيام معظم الأنظمة التي تدعم الانتقال من العنوان الافتراضي إلى العنوان الثانوي بدل التبديل بين الملفات تكون الجدولة متوسطة المدى فيها تؤدي دور الجدولة طويلة المدى.

نوره المحيسن – نوف المانع – سارة الشثري

المراجع:

Operating System Concepts
http://en.wikipedia.org/wiki/Scheduling
http://en.wikipedia.org/wiki/Scheduling_%28computing%29
http://en.wikipedia.org/wiki/Ready_queue

الجدولة الإجهاضية (Preemptive Scheduling) :

الجدولة القابلة للإجهاض(preemptive scheduling):

وهذا يسمح بمقاطعة العملية التي يتم تنفيذها حتى وإن كانت في منتصف التنفيذ ليأخذ التحكم بالمعالج منها ويعطيه للعملية الأخرى .

الجدولة غير القابلة للإجهاض (non-preemptive scheduling) :

وهذا يضمن للعملية أنها لن تترك المعالج حتى ينتهي وقت تنفيذها الحالي.

قرار جدولة وحدة المعالجة المركزية قد يحدث تحت الظروف الأربعة التالية :

1- عندما تنتقل عملية (Process) من حالة التشغيل إلى حالة الانتظار (مثال: نتيجة طلب إدخال/إخراج , أو انتظار لإنهاء أحد عمليات الطفل (child)).

2- عندما تنتقل عملية من حالة التشغيل(running state) إلى الحالة الجاهزة(ready state) (مثال:عندما ينتهي الوقت (time out) فإن وحدة المعالجة المركزية تنهي العملية).

3- عندما تنتقل العملية من حالة الانتظار إلى الحالة الجاهزة (مثال: المقاطعة).

4- عندما تنتهي العملية (Terminate).

عندما تحدث الجدولة تحت الظرف 1 أو 4 نقول أن مخطط الجدولة متعاون(non-preemptive) وفي الحالات الأخرى نقول عنه إجهاضي (preemptive).

تحت الجدولة التعاونية (non-preemptive) عندما تخصص وحدة المعالجة المركزية (CPU) لعملية فإن هده العملية تبقى في وحدة المعالجة المركزية (CPU) حتى تخرج بنفسها إما لأنها انتهت أو لانتقالها لحالة الانتظار .

الجدولة التعاونية (non-preemptive) هي الطريقة الوحيدة التي تستخدم برامج أجهزة معينة , لأنها لا تتطلب أجهزة خاصة (مثال :المؤقت Timer)، قلب النظام (Kernel) ليس لديه أي صلاحية لإخراج العملية من وحدة المعالجة المركزية حتى تنهي تنفيذها (مثل DOS)

جدولة الإجهاض (preemptive) بسب حاجة معينة يقوم قلب النظام (Kernel) بإجهاض العملية الموجودة في وحدة المعالجة المركزية (CPU) وإخراجها وإدخال عملية أخرى غيرها .
ولولا هذه الطريقة لما كان لدينا المعالجة المتعددة (Multiprocessing)، وهذه الطريقة أغلب أنظمة التشغيل تعمل عليها .

هند المطيري – فايزة الشمري **المراجع:** Operating System Concepts

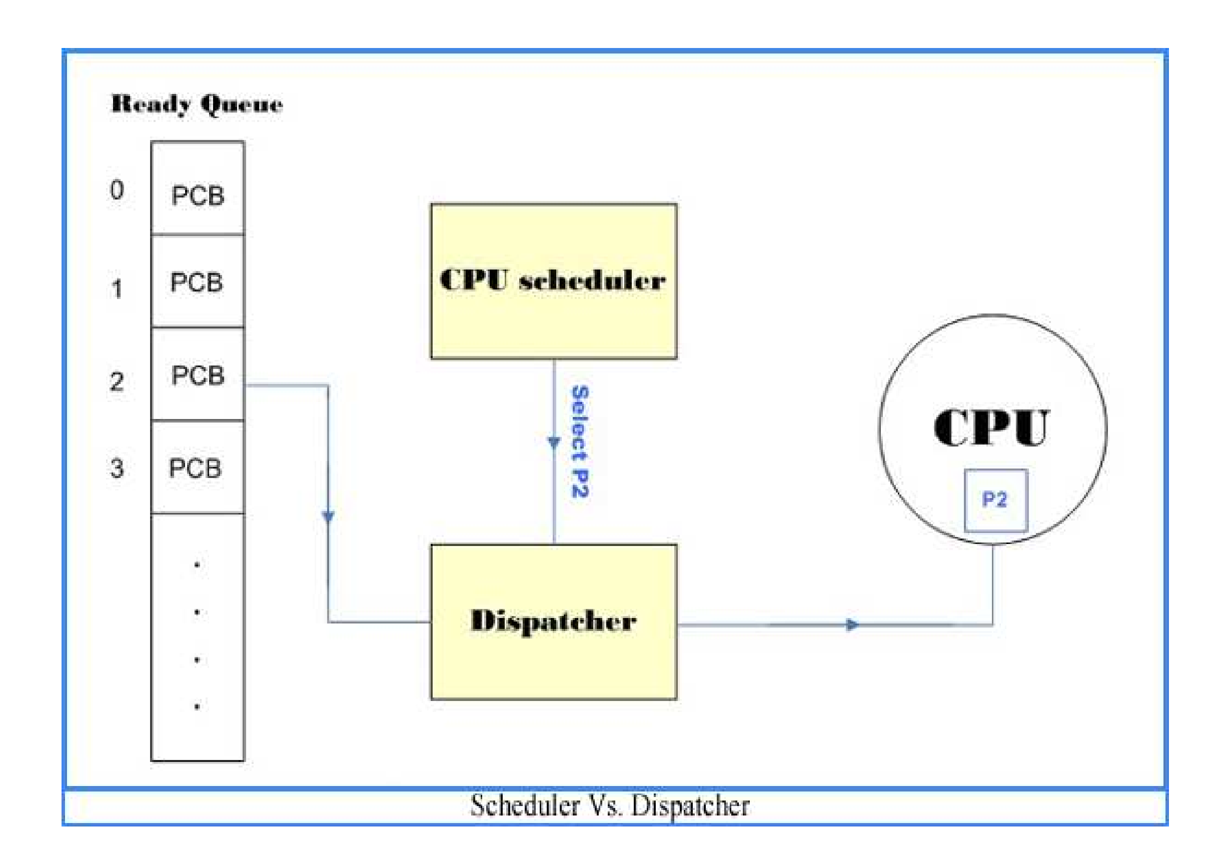
ما هو الdispatcher؟

الdispatcher بالمعنى اللغوي هو الشخص "المنفّذ " أو "المنجز" لمهمة ما, أما في نظام التشغيل فان الdispatcher يقوم بالمشاركة في عملية جدولة المعالج(CPU-scheduling), فهو عبارة عن وحدة(module) تقوم بتنفيذ القرارات التي يصدرها المجدول قصير المدى (short -term scheduler) .

في جدولة العمليات لدينا جزأين مهمين يقومان بهذه العملية, هما المجدول (CPU scheduler) , والمرسل (dispatcher), **فما هو الفرق بينهما؟**
يقوم المنسق أو المجدول(CPU scheduler) باختيار عملية من العمليات الجاهزة في الذاكرة(ready queue) ليتم إدخالها في وحدة المعالجة المركزية (CPU)، أي أن وظيفته هنا تكمن في اتخاذ القرار.
أما المرسل(Dispatcher) , فوظيفته تكمن في الجزء العملي, أي أنه هو الذي يقوم فعلياً بالتنفيذ وجلب العملية- التي اختارها المجدول- وإرسالها إلى وحدة المعالجة المركزية (CPU) لتنفيذها.

ويتضمن عمله عدة أمور
1- التبديل من عملية إلى عملية (context switch).
2- التبديل إلى طبقة المستخدم.
3- القفز إلى المكان المناسب في برامج المستخدم لإعادة تشغيل البرامج.

وغيرها من العمليات التي تحتاج لوقت يطلق عليه وقت المرســـل (dispatcher latency) وهو الوقت الذي يقضيه و يستغرقه المرسل لإيقاف أحد العمليات والبدء في تنفيذ أخرى الـــصورة التالية توضح لنا الفرق بينهما:



Scheduler Vs. Dispatcher

سمية باعطوة – قطر الندى السماعيل – حليمة حكمي

المراجع:

Operating System Concepts

معايير الجدولة (Scheduling Criteria) :

عملية الجدولة من أهم الخصائص في تشغيل العمليات حيث ينظم دخول العمليات المراد تنفيذها إلى وحدة المعالجة المركزية (CPU) وتعتمد هذه العملية على العديد من المعايير التي تحدد من هي العملية التي يجب تنفيذها و من أهمها:

1- استغلال وحده المعالجة المركزية (CPU utilization)

استغلال كل وقت وحدة المعالجة المركزية(CPU) في تنفيذ العمليات، أي أن تكون وحده المعالجة المركزية مشغولة بقدر الإمكان ليتم استغلالها الاستغلال الأمثل

وعادة يمثل بنسبة مئوية يمكن حسابها باستخدام أحد القانونين :

أ- نسبة استغلال المعالج = (وقت المعالج الكلي – الوقت الذي قضاه فارغا)/ (وقت المعالج الكلي) * 100

ب- نسبة استغلال المعالج = (وقت التنفيذ للعمليات الكلي)/ (وقت العمليات الكلي + الوقت المستغرق في تبديل المحتوى)* 100

ومما يؤثر على هذا العامل حقيقة هو عدد مرات التبديل التي تتم فكلما زاد هذا العدد كلما قل استغلال المعالج وهذا منطقي جدا أليس كذلك ؟ !

ما نريده نحن ونصبو إليه هو أعلى استغلال ممكن للمعالج إذ أننا لا نريد الآن شغل وقت فراغ المعالج فقط بل شغله بما ينفع أيضا فلا نريد تضييع وقته في عمليات التبديل.

2- الإنتاجية أو سعة المعالجة (Throughput)

عدد العمليات التي يتم تنفيذها في الوحدة الزمنية الواحدة

3- الوقت الدوري (Turnaround time)

الوقت اللازم لإنهاء تنفيذ عمليه محدده، و هو مجموع الفترات التي أمضاها في

أ- الانتظار قبل الدخول إلى الذاكرة

ب- الانتظار في طابور الاستعداد (ready queue)

ت- التنفيذ على وحدة المعالجة المركزية

ث- تنفيذ عمليات الإدخال والإخراج

4- وقت الانتظار (Waiting time)

هو الوقت الذي تستغرقه العملية في الانتظار داخل مصفوفة الانتظار(ready queue) قبل دخولها إلى وحدة المعالجة المركزية (CPU)

5- زمن الاستجابة (Response time)

الوقت من أمر تنفيذ العملية حتى ظهور أول نتيجة لها، يستخدم هذا عادة في الأنظمة التفاعلية interactive) system) التي يكون بها المستخدم طرفا.

وما نطمح إليه هو الحصول عليه هو استغلال للمعالج بأقصى حد ممكن maximum CPU) utilization) وأعلى نسبة من العمليات التي تكتمل في وحدة زمنية(Throughput)

كما نرغب في الحصول على أقل وقت انتظار(Waiting time) وأقل زمن استجابة (Response time) وأقل زمن دوري للعملية (Turnaround time)

إن هذه المعايير تتضارب فيما بينها

فزيادة استغلال المعالج (CPU utilization) تعني أن نقلل من تبديل العمليات context) Switching)حيث أنها تستغرق وقتاً (وفي هذه الحالة يكون المعالج غير مستغل) ؛ فلا تكون بصورة دورية ليتسنى للعملية داخل المعالج استغلال وحدة المعالجة المركزية (CPU) ، ولكن ذلك يعني زيادة زمن الاستجابة (Response time) للعمليات في طابور الاستعداد (Ready Queue) .

وكذلك تقليل معدل الزمن الدوري (Average turnaround time) للعمليات يتطلب تنفيذ العمليات الأقصر أولاً مما يسبب مجاعة للعمليات المتطلبة فترة زمنية طويلة، فيزداد زمن انتظارها أي سيرتفع أقصى زمن انتظار(maximum waiting time) ومن ثم سيزداد معدل زمن الانتظار .

إن المطلوب هو التوازن على كل حال؛ وما يُرَكَز عليه في تقييم الخوارزميات سيكون معدل زمـــن الانتظار (average waiting time) .

فاطمة الفرج – فايزة الشمري – نوره الخالدي – أنفال العواجي – خلود الرومي

المراجع:

Operating System Concepts

خوارزميات الجدولة Scheduling Algorithms :

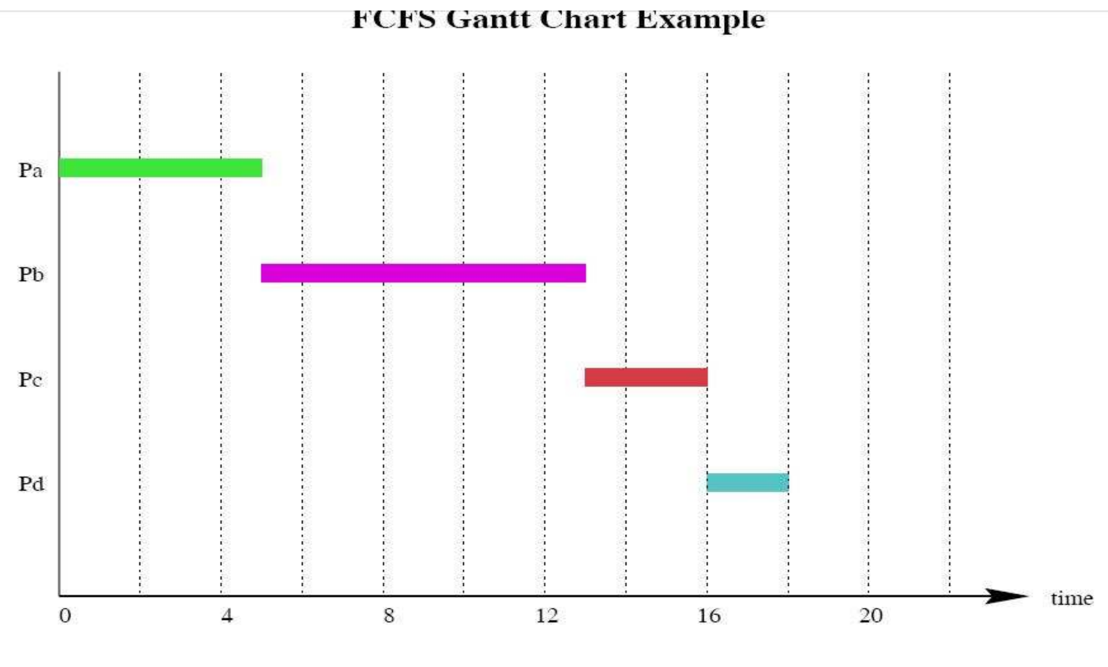
من يأتي أولاً يخدم أولاً (First-come First-served (FCFS-FIFO :

فلسفة هذه الطريقة تعتمد على من يصل أولاً من العمليات هو من يدخل أولاً إلى وحـــدة المعالجـــة المركزية (CPU) ،وهي تعتبر غير قابلة للإجهـــاض (Non-preemptive) أي أن هـــذه العملية لا تخرج من وحدة المعالجة المركزية إلا بعد انتهائها من التنفيذ ولا يتـــدخل لـــب النظـــام (kernel) في إجهاض العملية .

خصائص الخوارزمية :

1. أبسط خوارزميات جدولة المهام على الإطلاق .
2. معدل وقت الانتظار فيها ليس بالضرورة أن يكون الأقصر .
3. الأداء (performance) هنا يعيبه عدم الاستغلال الأمثل للمعـــالج وهـــذا ســـببه (convoy effect)ونعني به أن هناك عمليات قصيرة ويمكن إنجازها بسرعة ولكنـــها رغم ذلك تضطر للانتظار بسبب وجود عمليات أطول منها في طور التنفيذ .
4. يعتبر غير ملائم أبداً للاستخدام في الأنظمة التفاعلية (interactive system) وهذا العيب ناشئ وبشكل واضح عن كون هذه الخوارزمية غـــير قابلـــة للإجهـــاض (non preemptive scheduling)

مثال 1:



Initial ready queue: Pa = 5 Pb = 8 Pc = 3
Thread Pd (=2) "arrives" at time 5

مثال **2**:

(Process) العملية	*(Burst time)وقت التنفيذ*
P1	24
P2	3
P3	3

زمن الوصول لجميع العمليات هو t0 و ترتيب وصول العمليات هو P1,P2,P3

Gantt chart:

0-23	24-26	27-30
P1	P2	P3

معدل وقت الانتظار(Average waiting time) = (0+24+27)/3= 17

لو كان ترتيب وصول العمليات P2,P3,P1 :

0-2	3-5	6-30
P2	P3	P1

معدل وقت الانتظار(Average waiting time) = (0+3+6)/3= 3

العملية الأقصر أولاً Shortest-Job-First (SJR) :

تأتي كل عملية مصاحبة للوقت الذي تحتاجه للتنفيذ ويتم اختيار العملية ذات الوقت القصير.

وينقسم إلى قسمان:

1. **غير القابلة للإجهاض (Non-preemptive)** : يتم اختيار العملية ذات الوقت الأقصر من مصفوفة الانتظار و لا تخرج من وحده المعالجة المركزية إلا بعد انتهاء وقت تنفيذها.

2. **القابلة للإجهاض (Preemptive)**: أي انه عند وصول عملية جديدة ووقت تنفيذها اقصر من الوقت المتبقي لتنفيذ العملية الحالية فيتم إجهاض العملية الحالية وإدخال العملية الجديدة إلى وحدة المعالجة المركزية، تعتبر هذه الطريقة الأمثل لأنها تعطي اقل قيمة لمتوسط وقت الانتظار لمجموعة من العمليات ولكن لا تعطي تقديرات دقيقة للوقت الذي تنفذ فيه العملية .

كيف يتنبأ بوقت العملية قبل حدوثها؟

يحاول الجدول الزمني التنبؤ بوقت تنفيذ العملية استنادا إلى تاريخ تنفيذ العملية السابق وإذا كانت العملية جديدة فيسجل أول وقت لها في التنفيذ .

ويستخدم هذه المعادلة لتحديد زمن تنفيذ العملية:

$$t(n+1) = w * t(n) + (1 - w) * T(n)$$

حيث:

$t(n+1)$ وقت العملية القادم

$t(n)$ وقت العملية الحالي

T(n) متوسط وقت العمليات السابقة

W وقت الانتظار وغالبا ما يتراوح قيمته بين 0 <= w <= 10

أولوية الجدولة:

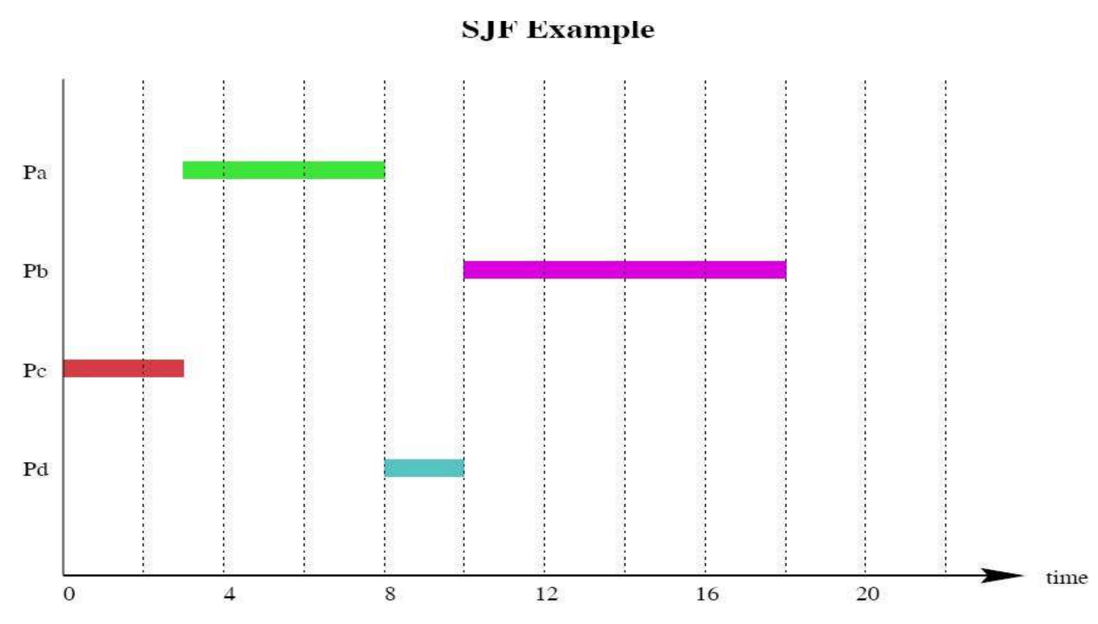
الأولوية هي عدد صحيح يرتبط مع كل عملية وإذا كان العدد مرتفع فان الأولوية تكون منخفضة والعكس صحيح (الأعداد الصغيرة تكون مرتفعة الأولوية في اليونيكس ولكن منخفضة في الجافا) .

ولكن هناك مشكلة :وهي إن العمليات ذات الأولوية المنخفضة لا تنفذ أبدا (مجاعة starvation) ، والحل: إن العمليات كلما زاد وقتها في الانتظار كلما زادت أولويتها حتى لا تهمل مع الوقت ولا يتم تنفيذها.

خصائص الخوارزمية :

1. صعبة البرمجة .
2. الخوارزمية التي تعطي أقل معدل انتظار على الإطلاق.

مثال 1:



مثال 2 (غير القابلة للإجهاض (Non-preemptive)):

(Process) العملية	***(Burst time)وقت التنفيذ***
P1	6
P2	8
P3	7
P4	3

Gantt chart:

0-2	3-8	9-15	16-24
P4	P1	P3	P2

معدل وقت الانتظار(Average waiting time) = (0+3+9+16)/4= 7

مثال 3 (القابلة للإجهاض (preemptive)):

العمليـــــــــــــــــة (Process)	***زمن الوصول (Arrival time)***	***وقـــت التنفيـــذ (Burst time)***
P1	0	8
P2	1	4
P3	2	9
P4	3	5

Gantt chart:

0	1-4	5-9	10-16	17-25
P1	P2	P4	P1	P3

معدل وقت الانتظار(Average waiting time) = ((10-1)+(1-1)+(17-2)+(5-3)) / 4 = 6.5

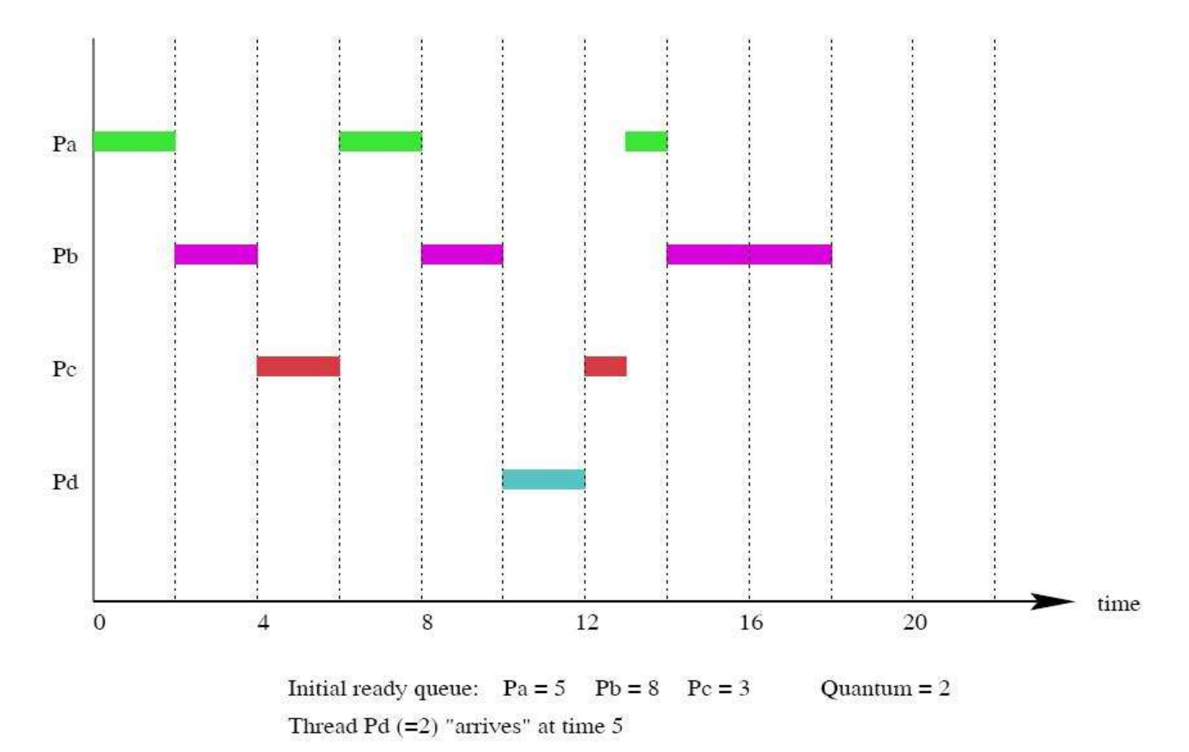
راوند روبن (Round Robin (RR) :

- هذه الخاصية تعطي لكل عملية وقت محدد من الـــزمن للتنفيـــذ داخـــل وحـــده المعالجـــة المركزية (CPU) ويسمى هذا الوقت (time quantum).
- يتراوح الوقت المحدد (time quantum) بـــين 10 -100 milliseconds وبعد انتهاء هذا الوقت تجهض العملية وتدخل في نهاية مصفوفة الانتظار .
- إذا كان هناك عدد من العمليـــات في مـــصفوفة الانتظـــار والوقـــت المحـــدد (time quantum) هو q فان كل عملية تأخذ 1\عدد العمليات
- لا يوجد عملية تأخذ أكثر من متوسط الوقت (1 - عدد العمليات) * الوقت المحدد.
- هذه الخاصية تشبه القادم أولاً يخدم أولاً (FCFS) ولكن تختلف عنها إن هنـــاك وقـــت محدد لتنفيذ العملية داخل وحدة المعالجة المركزية (CPU) .

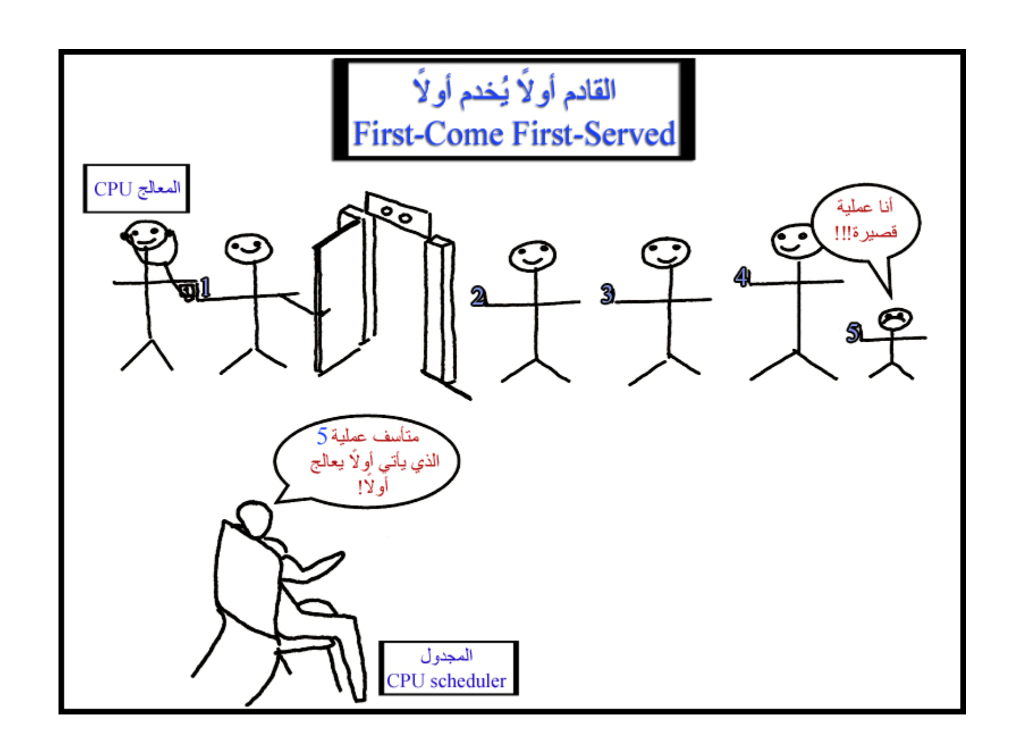
ملاحظات:

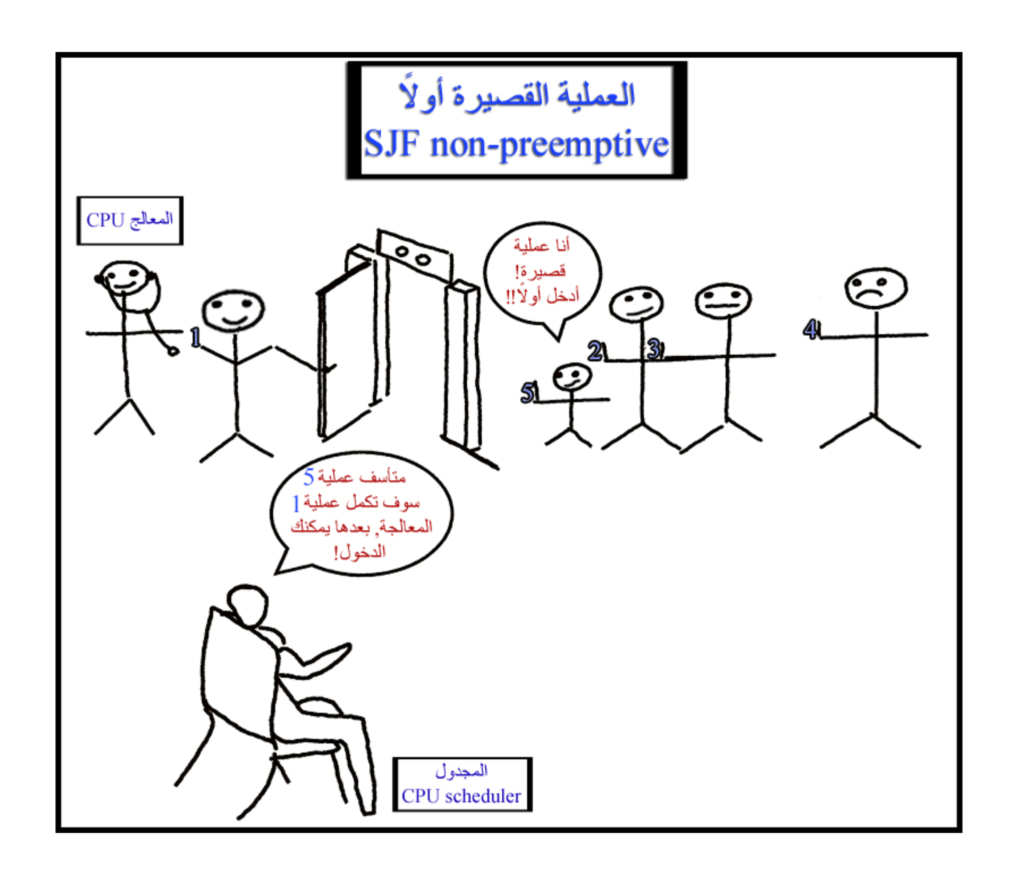
- إذا كانت قيمة وقت تنفيذ العملية اقل من الوقت المحدد فانه يأخذ وقت العملية
- في حالة إذا ضاعفنا الوقت المحدد (time quantum) فإننا سنـــصل إلى خوارزميـــة القادم أولاً يخدم أولاً (FCFS)
- وإذا قللنا الوقت المحدد (time quantum) فسيضيع وقت وحده المعالجة المركزية في التحويل (context switch)

مثال1:

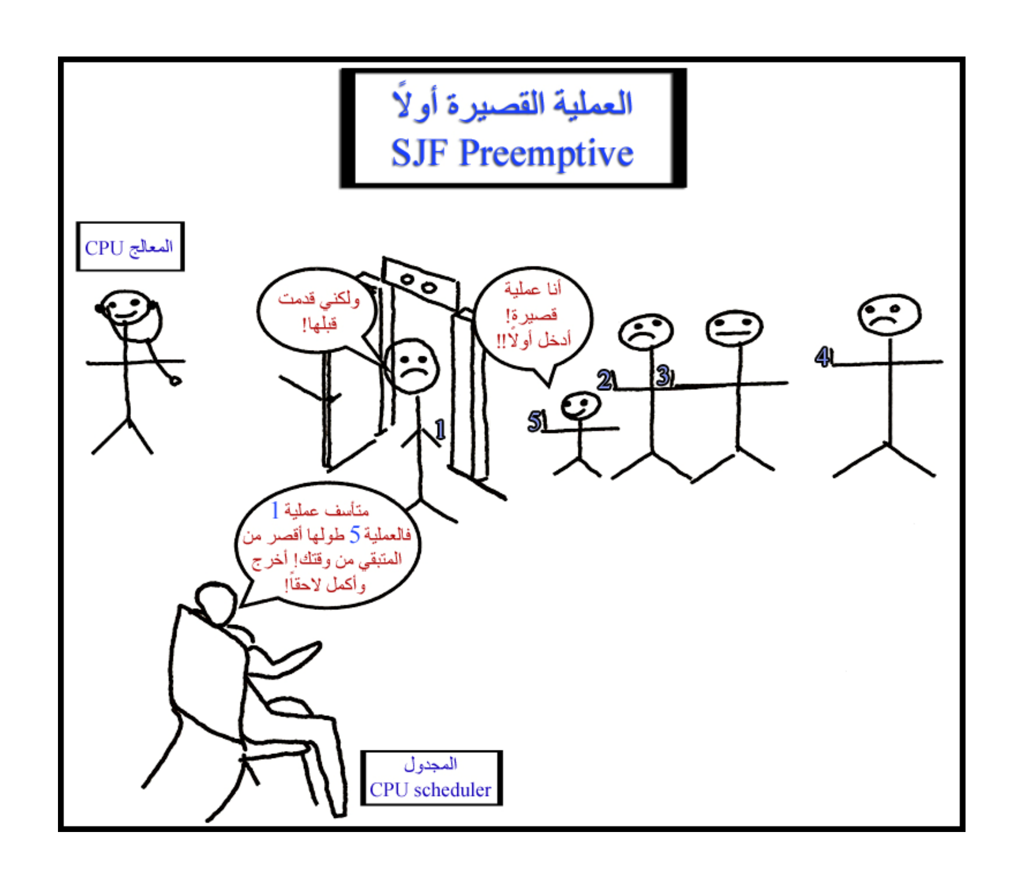


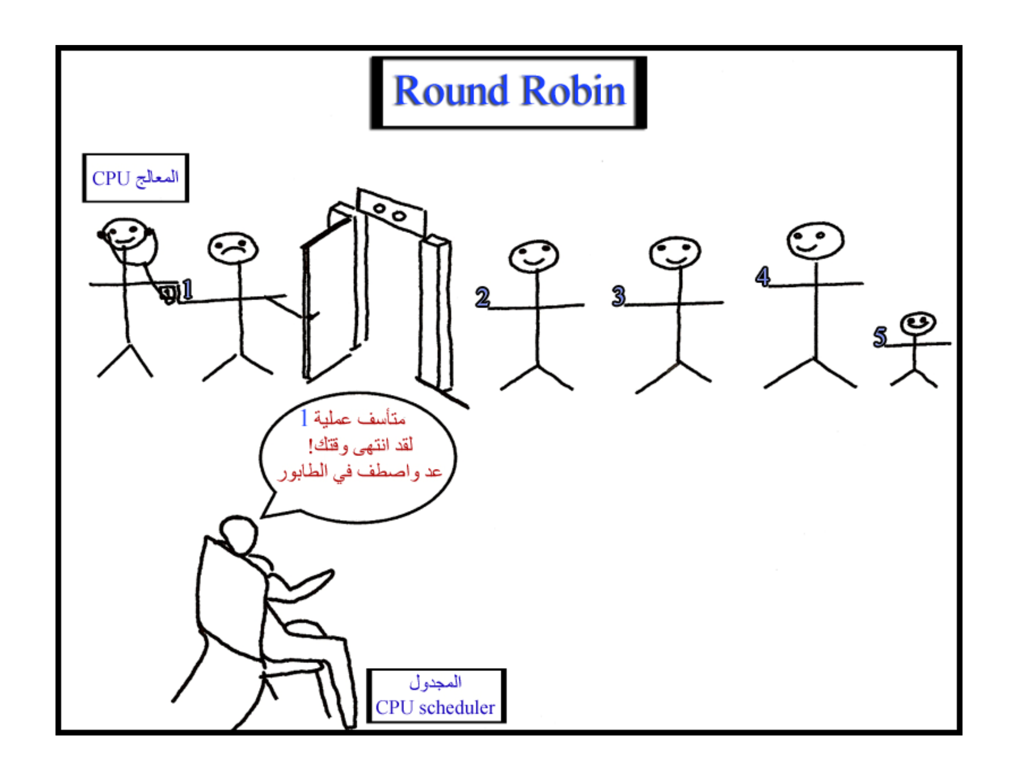
رسومات تبين طريقة كل خوارزمية:





العملية القصيرة أولاً
SJF non-preemptive
المعالج CPU
أنا عملية قصيرة! أدخل أولاً!!
1
2
3
4
5
متأسف عملية 5 سوف تكمل عملية 1 المعالجة, بعدها يمكنك الدخول!
المجدول
CPU scheduler


العملية القصيرة أولاً
SJF Preemptive
المعالج CPU
ولكني قدمت قبلها!
أنا عملية قصيرة! أدخل أولاً!!
1
2
3
4
5
متأسف عملية 1 فالعملية 5 طولها أقصر من المتبقي من وقتك! أخرج وأكمل لاحقاً!
المجدول
CPU scheduler


Round Robin
المعالج CPU
1
2
3
4
5
متأسف عملية 1 لقد انتهى وقتك! عد واصطف في الطابور
المجدول
CPU scheduler

طابور متعدد المستويات (Multilevel Queue):

تقوم هذه الخوارزمية بتقسيم طابور الانتظار(ready queue) إلى عدة طوابير (queues) مختلفة , ويتم إسناد المهمة إلى الــطابور المناسب بناء على عدة معاير منها:

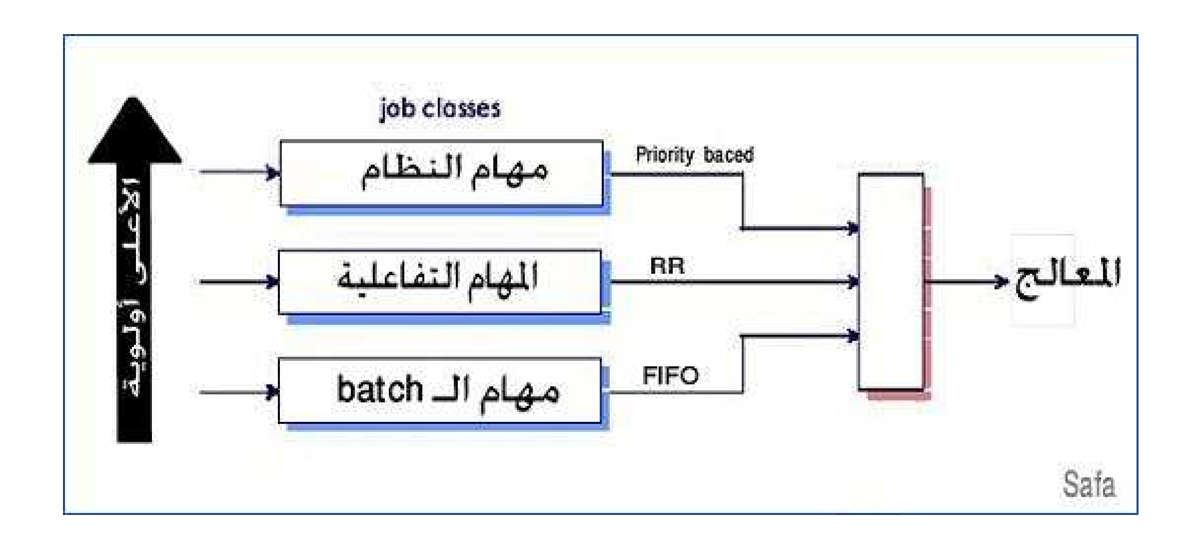
- حجم الذاكرة
- أولوية هذه المهمة
- نوع المهمة

ويكون لكل طابور (queue) خوارزمية يتبعها لجدولة المهام المسندة إليه.
وفي حالــة إســناد المهمــة إلى طــابور (queue) معــين لا تــستطيع المهمــة التحــول إلى طابور (queue) آخر من طوابير الانتظار (ready queues) (بخلاف طــابور متعــدد المستويات مع إمكانية التنقل multilevel feedback)

أما خوارزمية الجدولة بين هذه الطوابير (لاحظي بين الــطوابير وليس المهمات في داخل كل طابور) فتتبع إحدى الطريقتين التاليتين:

1. **الأولوية الثابتة**: بحيث أنه يسند لكل طابور أولوية ثابتة فيبدأ بتنفيــذ العمليــات التابعــة للــطابور صاحب الأولوية الأعلى وإذا انتهى من جميع عملياتها ينتقل إلى الــطابور الــذي يليه بالأولوية وهكذا دواليك، ولكن هذه الطريقة تسبب مجاعة أي أن العمليــات التابعــة للــطابور صاحب الأولوية الدنيا تتأخر كثيرا بالتنفيذ حيث تنتظــر إلى أن تنتــهي جمــع العمليات التابعة للــطوابير الأعلى أولوية.

2. **طريقة تقسيم الوقت (time slice)** : حيث يحدد مقدار ثابــت (مــثلا 5 ثــوان) فتعطى الــطابور الأول هذا المقدار من وقت المعالج ثم ينتقل إلى الــطابور الــذي يليــه ويعطى هذا المقدار المحدد وهكذا دواليك، أي أن وقت المعالج يقسم بين هذه الــطوابير.



طابور متعدد المستويات مع إمكانية التنقل (Multilevel Feedback Queue):

هي تطابق فكرة طابور متعدد المستويات (multilevel queue) أي أن طـــابور الانتظـــار (ready queue) مقسم إلى عدة طوابير إلا أن المهمة تستطيع الانتقال من طابور إلى آخر.
فيكون النظام عبارة عن مجموعة من طوابير القادم أولاً يخدم أولاً (FIFO queues) وليست واحدة فقط (مستويات عدة) بحيث تعطى لكل طابور درجة أولوية محددة كما أن كل طابور يتبع خوارزمية معينة للجدولة.

ولكي تنفذ عملية يجب إتباع الآتي:

عندما تبدأ عملية جديدة فإن الــنظام يقوم بإدخالها للطابور (queue) الأعلى أولوية ولــتكن Q0ويتم تنفيذ العمليات فيQ0 بالترتيب ويحدد لكل عملية وقت معين بعد انتهاء الوقت إما أن تنتهي. و إلا يتم نقل المهمة لــ Q1 والتي تعتبر أقل أولوية منQ0 .
وبذلك فإن العمليات القصيرة نسبياً لن تأخذ وقتا طويلاً حتى تنتهي، حيث إنه من الممكن أن تنتهي وهي في Q0 أو Q1 أو Q2 لكن العمليات الطويلة والتي يمر عليها الدور أكثر مــن مــرة و لم تنتهي فإنها قد تصل إلى آخر طابور (last queue) بدون انتهائها و بدون أن تــؤثر علــى الــمهمات القصيرة، أي أنها تمنع حصول المجاعة وهذا يعد من مميزات هذه الطريقة

وتحتاج هذه الطريقة إلى عدد من المتغيرات:

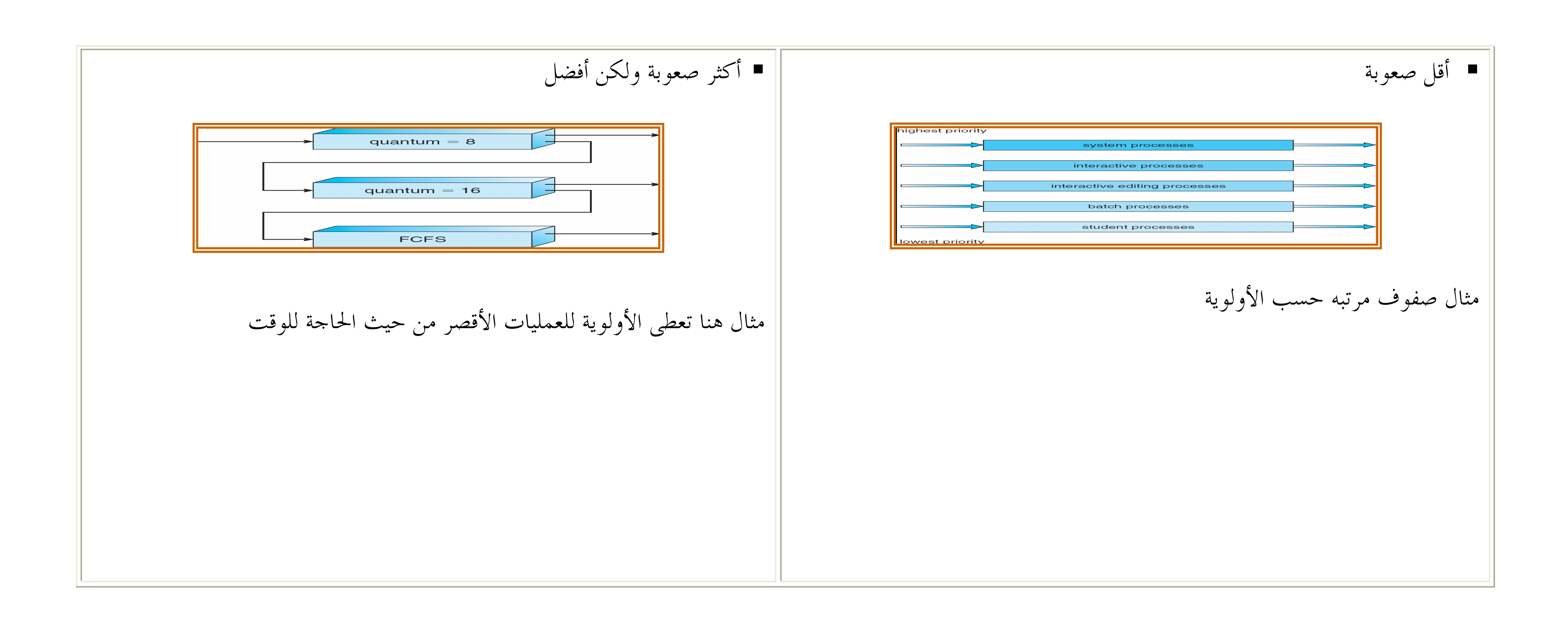
- عدد الصفوف
- طريقة جدولة كل صف
- متى ننقل العملية إلى مستوى أولوية أقل
- اختيار الصف التي تدخل فيه العملية

مقارنة بين خوارزميات الجدولة

القادم أولاً يخدم أولاً (First come first served)		العملية الأقصر أولاً (Shortest-job-first scheduling)		جدولة راوند روبن (Round-Robin)	طابور متعدد المستويات مع إمكانية التنقل (Multilevel feedback queue)	
مميزات	**عيوب**	**مميزات**	**عيوب**	**مميزات**	**مميزات**	**عيوب**
▪ بسيط ▪ قلة العمل الذي فوق القدرة ▪ لا تجويع للعمليات ▪ سهل الكتابة و الفهم	▪ إعطاء وقت استجابة قليل للعمليات التفاعلية ▪ وقت الانتظار له غالبا يكون طويل	▪ أفضل خوارزمية من حيث وقت الاستجابة	▪ احتمال حصول مجاعة ▪ CPU burst يتوقف على وقت العملية	▪ لا تجويع للعمليات ▪ انخفاض مدة الانتظار ▪ وقت استجابة جيد للعمليات التفاعلية	▪ مرنة ▪ أوقات استجابة جيدة للعمليات التفاعلية	▪ CPU burst يتوقف على وقت العملية

طابور متعدد المستويات (Multilevel queue)	طابور متعدد المستويات مع إمكانية التنقل (Multilevel feedback queue)
▪ الصف عديد الطبقات ▪ تدخل العملية إلى صف واحد اعتمادا على خصائصها ▪ لا تغير العملية الصف اللي دخلته من البداية ▪ يوجد مشكلة الظلم لبعض العمليات ▪ العوامل التي تحتاج تحديدها أقل، تحتاج إلى ▪ الخاصية التي تحدد للعملية الدخول في أي صف ▪ الطريقة المتبعة داخل الصف الواحد ▪ الطريقة المتبعة للتنظيم بين الصفوف	▪ الصف عديد الطبقات مع التقرير الرجعي ▪ تدخل العملية إلى أول صف واختيار الصف اعتمادا على الطريقة المتبعة ▪ يتم التنقل بين الصفوف حسب طريقه متبعه ومحدده ▪ تم الانتهاء من مشكلة الظلم ▪ العوامل التي تحتاج تحديدها أكثر، تحتاج إلى ▪ الخاصية التي تحدد للعملية الدخول في أي صف ▪ الطريقة المتبعة داخل الصف الواحد ▪ الطريقة المتبعة للتنظيم بين الصفوف ▪ كيفية الانتقال بين صف وآخر إما بزيادة الأولوية أو إنزالها

مقارنة بين Multilevel Queue & Multilevel Feedback Queue



فايزة الشمري – فاطمة الفرج – صفا البلاع – سماح المطلق – ابتهال العتيبي – الجوهرة الرشيد – قطر الندى السماعيل– ديمة السويد - خديجة أخرفي – نوف العجمي

المراجع:

Operating System Concepts

https://www.it.uu.se/edu/course/homepage/oskomp/vt07/lectures/scheduling_algorithms

http://www.personal.kent.edu/~rmuhamma/OpSystems/Myos/multQueue.htm

http://en.wikipedia.org/wiki/Multilevel_queue

http://cs.wwc.edu/~aabyan/352/Scheduling.html

http://www.cs.jhu.edu/~yairamir/cs418/os2/tsld039.htm

http://www.mines.edu/Academic/courses/math_cs/macs442/resources/S9schedul-Part2.pdf

http://en.wikipedia.org/wiki/Multilevel_Feedback_Queue

www.student.cs.uwaterloo.ca/~cs350/S04/notes/sched.pdf

جدولة المعالج المتعددة (Multiple-Processor Scheduling)

كنا نتحدث سابقا عن معالج واحد (one CPU) وكيفية عمل الجدولة له، والسؤال هنا ماذا لو كان لدينا أكثر من معالج (multiple CPUs) ؟!

بالطبع سيصبح الوضع أصعب في حال وجود أكثر من وحدة معالجة مركزية واحدة (many CPUs) لأننا نستخدم مزيج من الخوارزميات .

وهناك قواعد مختلفة للمعالجات المتجانسة (homogeneous processor) و الغير متجانسة (heterogeneous processor)

وتعني كل منهما ما يلي:

1. **المعالجات المتجانسة (Homogeneous)**: المعالجات لها نفس الخصائص تماما , مثل تطابق عدد السجلات(register), ويتواجدوا على نفس اللوحة الأم (motherboard) .وغالبا ما يكونوا من نفس النوع وبالطبع التجانس أفضل.

2. **المعالجات الغير متجانسة (Heterogeneous)**: أي المعالجات ليست متطابقة .

وفي حال وجود أكثر من معالج لابد أن ننتبه للخصائص التالية:

1. **الاشتراك في الحمل أو التحميل (Load sharing)** : أي الاشتراك في توزيع العمل ولابد أن تصبح المعالجات لها نفس فرص العمل ، ولابد أن تكون جدولة المعالج (cup schedule) قادرة على عمل توازن بين تلك المهام.

2. **متعدد المعالجة اللا متناظرة (Asymmetric multiprocessing)**: وهي أن كل قرارات الجدولة , وعمليات الإدخال والإخراج , ونشاطات النظام الأخرى يملكها معالج

واحد فقط ,ويسمى بالخادم وبقية المعالجات الأخرى تنفذ المطلوب منها ، هي تعتـــبر بـــسيطة فمعالج واحد فقط هو الذي يستطيع الوصول إلى تراكيب البيانات، حيث توجد نسخة واحدة فقط من نظام التشغيل على معالج (master) الذي يقوم بتنفيذ تعليمات نظام التشغيل، بينما تكون جميع المعالِجات الأخرى هي معالِجات تابعة (slaves) تنفذ فقط المهام الموكلة إليها من قبل المعالِج (master).

المشاكل التي تواجه هذا النوع:

1. تكمن في الحمل الزائد على المعالج (master)
2. كثرة الطلبات و الاستدعاءات حيث تظهر مشكلة ما يسمى بعنق الزجاجة

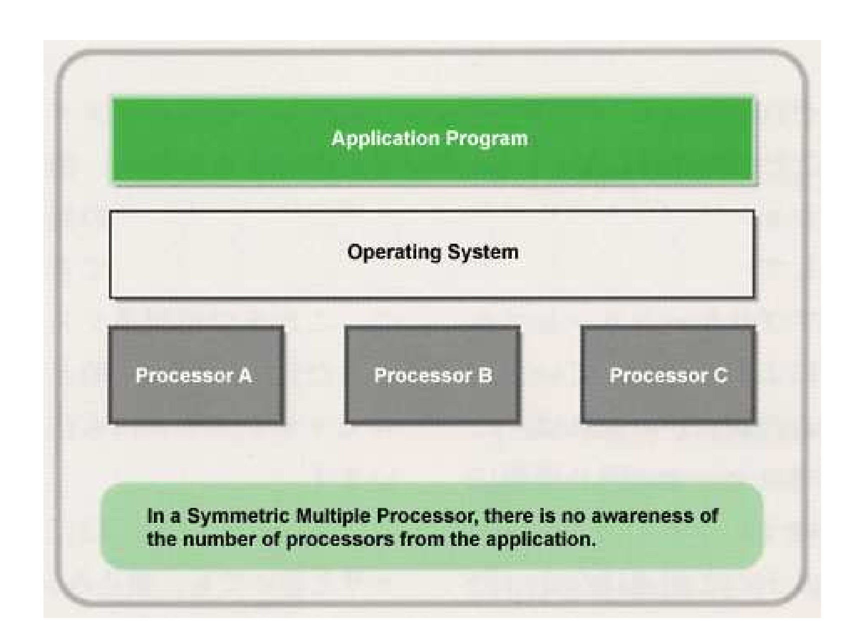
3. متعدد المعالجة المتماثلة ((Symmetric multiprocessing (SMP)) : كل معالج له جدولة خاصة فيه , بحيث أن كل معالج له قراراته وليس له علاقة بأي معـــالج آخـــر، حيث توضع نسخة واحدة من نظام التشغيل في ذاكرة مشتركة يمكن الوصول إليها من قبل باقي المعالِجات.

المشاكل التي تواجه هذا النوع:

1. عندما يطلب معالجان أو أكثر بنفس الوقت التعامل مع شفرة نظام التشغيل الموجودة في الذاكرة وهي من أهم المشاكل التي تواجه هذا النوع ، وغالباً ما تُحل هـــذه المـــشكلة بوضع ما يشبه القفل (معروف بـــ mutex) لتنظيم لائحـــة الوصـــول إلى نظـــام التشغيل في المرة الواحدة . فعندما يرغب معالج ما بالتعامل مع شفرة النظام يجـــب أن يحصل في البداية على الإذن بذلك (أي يمتلك القفل)فإذا كان القفل غير متاح حاليـــاً يجب على هذا المعالج الانتظار حتى تحرير القفل من المعالج الذي أغلقه .
2. تحقيق التزامن بين المعالِجات
3. كيفية منع الجمود (deadlocks)
4. تنظيم المشاركة الزمنية(time slice)

يوجد طريقتين لتنفيذ العمليات :

1. أما عن طريق طابور(queue) مشتركة لكل المعالجات
2. أو لكل معالج له طابور (queue) خاصة فيه



أفنان البحيري – عالية المصيريعي – منى البرية

المراجع:

Operating System Concepts

http://www.cse.sc.edu/

جدولة الخيط أو التجزئة Thread Scheduling

تجزئة المستخدم(user thread) نعمل لها تخطيط وربط بطبقة النظام (kernel thread) ذلك لأن نظام التشغيل هو من يعمل الجدولة على الخيوط أو التجزئة(thread).
والجدولة تكون على التجزئة أو الخيط (thread) وليس على المهمة (process) ولابد أن ترتبط التجزئة بالنظام (kernel thread).

وهناك نوعين من الجدولة على التجزئة أو الخيوط (threads) :

أولا: الجدولة المحلية Local Scheduling (process-contention-scope PCS)

وتعني أن مكتبة التجزئة (thread library) هي المسئولة عن عملية ربط التجزئة الموجودة في طبقة المستخدم بنظام التشغيل (kernel) عن طريق إقرار أي تجزئ يدخل على (LWP) , ذلك أن (LWP) محدودة، و تسمى جدولة محلية.
وفي لغة السي (c) نستخدم الأمر التالي لعمل ذلك : PTHREAD_SCOPE_PROCESS

ثانيا: الجدولة العالمية Global Scheduling (system-contention-scope SCS)

هنا القرار يكون للب النظام (kernel) حيث يقرر أي من التجزئات (kernel thread) يدخل على وحدة المعالجة المركز(CPU)لتتم معالجته.
ونستخدم الأمر التالي حتى تتم عملية الربط: PTHREAD_SCOPE_SYSTEM

منى البرية

المراجع:

Operating System Concepts
http://www.cse.sc.edu/

الفصل السادس:

الجمود

الجمود

(Deadlock)

تُعتبر فلسفة بناء نظم التشغيل من أجمل العلوم وأعقدها بنفس الوقت.. وعموما أنظمة التشغيل يواجهها مشاكل كثيرة جداً وقد لايستطيع أحد إحصاءها. يحاول كاتبوا نظام التشغيل تزويد النظام بحلول مسبقة للمشاكل المتوقع حدوثها. لكن بسبب كثرة وعشوائية العوامل التي يتعرض لها نظام التشغيل.يصعب التعرف بنوعية هذه المشاكل.

سأتطرق في هذا الموضوع لمشكلة أساسية في نظم التشغيل وهي الجمود (**Deadlock**)

في البداية لابد من معرفة أهداف كتابة هذا الموضوع:

1– التعرف على أهم المشاكل التي تواجهها أنظمة التشغيل.

2– لمعرفة مصطلح الجمود والأسباب المؤدية له.

3– لتطوير وصف الجمود الذي يمنع مجموعة العمليات المتلاقية من إكمال مهامهم.

4– لتقديم عدد من الطرق المختلفة لمنع أو تفادي الجمود في نظام الحاسب.

جمع وترتيب نورة قيسي (norah_gaissi@hotmail.com)

ماهو الـــجمود؟

هو منع العمليات من استخدام موارد النظام بشكل دائم بسبب تنافسها مع العمليات الأخرى على هذه الموارد أثناء عملية الاتصال بينهم.

```
P0          P1
wait(A);    wait(B);
wait(B);    wait(A);
```

سبب حدوث الجمود

عندما تكون هناك عمليات تنتظر تنفيذ مهام- قامت فيها عمليات سابقة- وهذه العمليات السابقة هي بدورها تنتظر انقضاء مهام أخرى.

لتبسيط هذه العملية نأخذ هذه الامثله :

المثال الأول:

لتكن هناك عمليتان س وص كلاهما تريدان طباعه مستند موجود على محرك الأقراص

1- س طلبت استخدام الطابعة وأخذت الموافقة

2- ص طلبت استخدام محرك الأقراص وأخذت الموافقة

3- س طلبت استخدام محرك الأقراص لكن طالبها رفض ريثما ينتهي ص من استخدامه

4- ص طلب استخدام الطابعة

هنا يحدث الجمود!!

المثال الثاني:

لدينا قلم و مسطره و هناك شخصين يريدون الرسم احدهم يحمل القلم والأخر المسطرة

في حاله ان الشخص الذي يحمل القلم أراد الحصول على المسطرة من الشخص الأخر و الشخص الذي يحمل المسطرة أراد القلم فإن الــجمود يظهر فيه هذه الحالة لأن كلا الطلبين لا يمكن تحقيقهم .

المثال الثالث:

الجسر...حيث انه يكون ذو اتجاه واحد..ولكن ماذا يحدث إذا خالفت سيارة ذلك وتعاكست سيارتان؟؟ سيتولد الجمود .

ونستطيع حله بأن تتراجع إحدى السيارات، أي أننا أجهضنا إحدى السيارات عن حركتها (preemption)، ولكن المشكلة في حالة العمليات أننا لا نستطيع أن نجهض العملية دائما...

المثال الرابع:

لدينا عمليتين وكل واحدة منهم تريد أن تطلب ملفين ، العملية الأولى طلبت الملف الأول والعملية الثانية طلبت الملف الثاني وبذلك تنتظر كل عملية الأخرى لتحرر الملف حتى يمكنها من طلبه..

<u>**الموارد (Resources)**</u>:

هي الأشياء التي تستخدمها العملية لتنفيذ مهامها قد تكون:

الأجزاء الصلبة من الحاسوب(الهاردوير) مثلا محرك الأقراص, الطابعة أو غيرهما.

أو قد تكون معلومات مثلا ملف أو غيره.

أي أن الموارد هي أي شي يستخدم لتنفيذ عمليه مخصصه يحجز لمدة زمنيه محدده أي إلى انقضاء تنفيذ المهمة.

هناك نوعان من الموارد:

1- يمكن احتكارها (Preemptable) **مثل** الذاكرة .

2- لا يمكن احتكارها (Non-Preemtable) **مثل** الطابعة.

الموارد التي يمكن احتكارها:

هي الموارد التي تعين لعملية ما إلى أن تنتهي من تنفيذ مهمتها ثم تعين إلى مهمة أخرى بدون أي تصادم.

الموارد التي لا يمكن احتكارها:

هي الموارد التي لا يمكن أن تؤخذ من عملية وتعين إلى أخرى بدون أن تسبب تصادم.

"عند ذكرنا لمثال الطابعة لا نقصد أن تعين إلى عمليه أخرى خلال تنفيذ مهامها, بل نظام التشغيل يعامل الطابعة كمورد يمكن احتلاله –المشغل يرتب المهام التي يجب على الطابعة تنفيذها

يحدث في الموارد التي لا يمكن احتلالها كما سنرى أن الجمود

التسلسل لاستخدام مورد هو:

1– طلب المورد:إذا كان المورد متوفر(أي لا تستخدمه عمليه أخرى) فإنه يمنح على الفور أو ينتظر إذا كان يستخدم في عملية أخرى.

2– استخدام المورد:في حال الحصول عليه.

3–تحرير المصدر:في حال الانتهاء من تنفيذ المهمة.

شروط حدوث الجمود

1:(Mutual exclusion)– منع التبادل (

فقط عملية واحده يمكنها أن تستخدم المورد. إذا كان المورد ممكن استخدامه بنفس الوقت لعدة عمليات فإن هذا لا يسبب Deadlock.

2– الاستخدام و الانتظار (Hold and Wait)

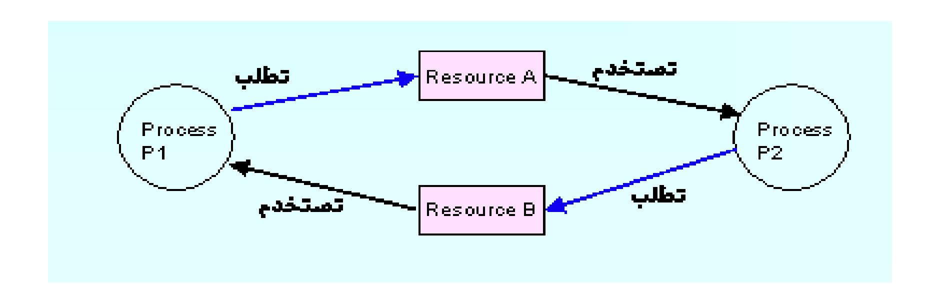
– أن تكون العمليات حاجزة مورد وتطلب آخر

3- عدم الإجهاض (no preemption):

أي أن العملية التي تحمل المورد لا يمكن إجهاضها و لابد أن تقوم هي بتحرير الموارد التي تحملها.

4- الانتظار الدائري (Circular wait):

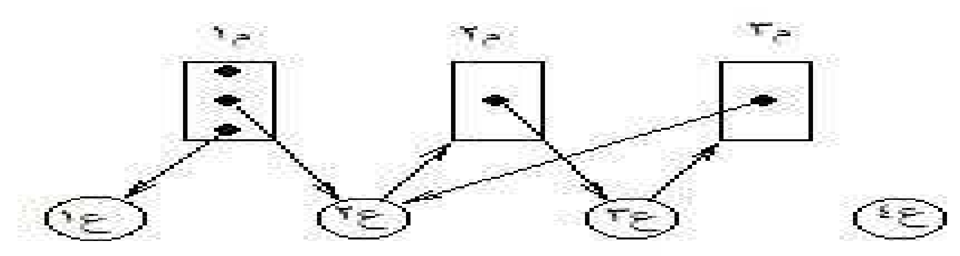
.بمعنى أن تتكون دائرة مغلقة من الإنتظارات تنشأ من احتكار عملية لمورد أو أكثر تنتظره عملية أخرى، وهذه العملية بدورها تنتظر عملية أخرى ....وهكذا.



** **علماً أن توافر هذه الشروط لا يضمن حدوث الجمود فهي ضرورية، ولكنها ليست كافية، لحدوث حالة الجمود.**

<u>**تمثيل الجمود:**</u>

(**Resource-allocation graph**)يمكننا استخدام رسم بياني يسمى **تعيين الموارد**



القمم تمثل الموارد و العمليات

مجموعة القمم تقسم إلى صنفين مختلفين :

1- ع= {ع1,ع2,ع3,ع4 } وهذه المجموعة تتكون من كل العمليات الفعالة في النظام.

2- م= {م1,م2،م3} وهذه المجموعة تتكون من أنواع الموارد في النظام .

.

سوف نستخدم **القمم المستطيلة** لتمثيل المورد **والنقاط الداخلية** لتمثيل الطلب على المورد وسوف نستخدم **القمم الدائرية** لتمثيل العملية.

الخطوط من العملية إلى المورد تمثل الطلب وتسمى **بخطوط الطلب**.

الخطوط من النقاط الداخلية في المورد إلى العملية تعني انه تم تعيين المورد لهذه العملية وتسمى **بخطوط التعيين**.

في الرسم البياني أعلاه نجد أن:**

هناك ثلاث طلبات على المورد م1, ونجد أن العمليتان ع1 وع2 تم تعيين المورد لهما.

ع2 يطلب مورد م2 لكن ينتظر الحصول عليه بعد أن ينتهي منه ع3.

ع3 يطلب م3 الذي يستخدمه ع2 هنا يحدث Deadlock.

الانتظار الدائري هنا يمكن تمثيله كالتالي:

ع2-> م2 -> ع3 -> م3 -> ع2

طرق التعامل مع الجمود:

1) تجاهل المشكلة (Ignore the problem):

هذه الإستراتيجية معقولة، الجمود من غير المحتمل حدوثه في أغلب الأحيان، فإن نظام التشغيل يمكن أن يعمل لسنوات دون حدوث جمود. نظام التشغيل إذا له نظام منع الجمود أو نظام كشف الجمود فإن هذا سيؤثر سلبيا على نظام التشغيل(يبطئ الجهاز).

2)كشف الجمود (Deadlock Detection):

دائما يلبي طلب المصدر إذا كان ممكن، ويراقب الجمود بشكل دوري، وإذا حصل جمود يقوم بمعالجته. إذا كان هناك instance لكل مصدر فإنه من المحتمل اكتشاف الجمود بواسطة الرسم البياني ونبحث عن وجود cycle .

3)منع الجمود (Deadlock Prevention):

يرفض أحد الشروط الأربعة للجمود.

4)تجنب الجمود (Deadlock Avoidance):

لا يلبي طلب المصدر إذا كان يمكن أن يؤدي إلى الجمود. فهذا يتفادى الجمود الذي قد يحدث للمصدر لاحقا.

**** الفرق بين منع الجمود deadlock prevention و تجنب الجمود deadlock avoidance** غير ملحوظ إلى حد ما . تشير إستراتيجية تجنب الجمود إلى أنه عندما يطلب المصدر فهو يمنح للعملية فقط إذا كان لا يؤدي إلى الجمود.

تتضمن إستراتيجية منع الجمود إلى تغيير القواعد بالتالي فإن العملية لن تتمكن من طلب المصدر الذي يمكن أن يؤدي إلى الجمود.

الحالة الآمنة:

هي الحالة الوحيدة التي تضمن أن كل العلميات باستطاعتها أن تقوم بالمهام التي يتوجب عليها القيام بها.

أما الحالة غير الآمنة لا تعطي مثل هذا الضمان.

نستطيع تفادي الجمود **Bankerباستخدام خوارزمية**

تنص هذه الخوارزمية انه لا تنفذ العملية إذا كانت ستضع النظام في حاله غير آمنه-أي أن ليس كل العمليات سوف تقول بمهامها.

يعلم مسبقا فيما يتعلق بالمورد التي سوف يتطلب أن يكون نظام التشغيل تفادي الجمود

تطلبها العمليات وتستخدم لفترة زمنية محدودة

شفاء الجمود:

توجد حلول عديدة لمعالجة الجمود بعد اكتشافه منها:أعلام المستخدم بوجود حالة الجمود وترك الخيار له لمعالجة الجمود أو ترك نظام التشغيل يعالج الجمود ويشفى منه أوتوماتيكيا.

وعندما يترك الخيار لنظام التشغيل الشفاء من الجمود فهناك خيارات لكسر الجمود:

1/إجهاض أحد المهام العالقة في الانتظار المسبب للجمود.

2/منع المهام العالقة في الجمود من بعض المصادر أو كلها.

أ/إجهاض المهمة:

لكسر الجمود عن طريق الإجهاض يسلك نظام التشغيل إحدى الطريقتين إما إجهاض جميع المهام العالقة في الجمود أو إجهاض أحد المهام العالقة في الجمود.

وفي جميع الحالات يسترد نظام التشغيل جميع المصادر التي لدى المهمة المجهضة.

1/إجهاض جميع المهام العالقة في الجمود:

وفي هذه الحالة سوف يكسر الجمود بشكل أكيد ولكن هذه العملية تعتبر مكلفة نظرا لان المهام المجهضة تضطر لإعادة العمل وحجز المصادر وإضاعة الكثير من الوقت.

2/ إجهاض أحد المهام التي تكسر حالة الجمود:

عندما يتم إجهاض أحد المهام يقوم نظام التشغيل باستدعاء خوارزمية اكتشاف الجمود التأكد من انتهاءه أ و وجوده وإذا كان الجمود لا يزال موجودا فلابد من إجهاض مهمة أخرى وإعادة العمل حتى ينتهي الجمود.

أن إجهاض المهمة ليس بالعمل السهل.حيث أن المهمة قد تكون في منتصف التعديل على ملف وإجهاضها يترك الملف في حالة غير صحيحة وكذلك أيضا إجهاض المهام التي تقوم بعملية طباعة فلابد من إعادة الطباعة قبل انجاز الطباعة التالية.

فلذلك عندما نقوم بإجهاض أحد المهام فلابد من اختيار المهمة ذات التكلفة الأقل.وتحديد المهمة ذات التكلفة الأقل ليس بالأمر السهل وهي تعتمد على سياسة نظام التشغيل وعلى الطريقة التي يتبعها لتحديد التكلفة.

ويعتمد تحديد تكلفة المهمة على:

1/أولوية المهمة. 2/مدى العمل الذي أنجزته. 3/كم عدد وما نوع المصادر التي لدى المهمة. 4/وكم عدد المهام التي أحتاج أن ينتهي تنفيذها. 5/كم سيحتاج من المصادر الإضافية. 6/أي المهام متفاعلة حاليا.

ب/التسابق على المصادر:

حيث تقوم المهمة بطلب المصادر التي تحتاجها فتحصل على البعض والبعض الآخر لا تستطيع الحصول عليه حتى لا تعلق المهمة في الجمود.

أما أذا كانت المصادر مع المهام العاقة في الجمود.فهناك 3 حالات:

1/اختيار الضحية:يعني أي المهمة سوف يتم إنهاؤها وتمتلك مصادرها.ولاختار المهمة الضحية يكون لدى نظام التشغيل أولويات للامتلاك حتى يحقق تقليل التكلفة.ويحدد عامل التكلفة بعدد المصادر وكمية العمل المنجزة لدىكل مهمة

2/إعادة تنفيذ المهمة:بعد أخذ المصادر من المهمة الضحية لا تستطيع المهمة المضي في التنفيذ بل يعاد التنفيذ إلى نقطة آمنة تستطيع المهمة التنفيذ منها بشكل صحيح.

3/التجويع:عندما نختار مهمة ضحية فكيف نتأكد انه لن يحصل لها تجويع أي لن تكون هي لضحية في كل مره يكون اختيار الضحية على أساس التكلفة القليلة وحتى لا تكون المهمة هي الضحية في كل مرة يقوم نظام التشغيل بإضافة عدد المرات التي كانت المهمة فيها هي الضحية إلى التكلفة وفي كل مرة يبحث فيها نظام التشغيل عن الضحية يقارن بين المهام العالقة في الجمود بعدد المرات التي حصل فيها للمهمة إعادة تنفيذ وهذا يعتبر من أكثر الحلول شيوعا.

*****أيضا من حلول الجمود**

1– الربط السابق لمصادر الحاسب قبل وقوع حالة الجمود, بحيث يتم جدولة العمليات المختلفة حسب حاجتها من مصدر الحاسب, وشرط تجنب حالة الجمود ,ولهذا الحل العيوب التالية:

أ– يحدث للمستخدم أن لا يعلم أوقات وفترات ارتباط كل مصدر من مصادر الحاسوب بالعمليات المختلفة, وذلك قبل بدء التنفيذ, وبالتالي يصعب تحديد عملية الربط المسبق لمصدر الحاسوب.

ب– عند ربط مصدر الحاسوب بعمل ما من الأعمال مسبقا , قد يتم حجز هذا المصدر فترة طويلة دون أن يكون لهذا المصدر أي عمل وظيفي يؤديه, وبذلك يؤخر تنفيذ الأعمال الأخرى.

ج– من العبث أن يتم ربط مصدر من مصادر الحاسوب مسبقا, خاصة إذا صغر احتمال استخدام ذلك المصدر بذلك العمل المرتبط به , كأن يتم ربط الطابعة مثلا مسبقا بعمل قد لا يحتاج أكثر من سطر واحد طباعة يتم طباعته في نهاية العمل.

2- ربط المصادر بالأعمال تحت شروط Constrained Allocationو ويتفرع من هذا الربط مايلي:

1- الربط القياسي Standard Allocation

2 - الربط المتحكم به Controlled Allocation

أي لا يتم ربط المصادر بالأعمال إلا وفق قيود محددة, تمنع حالة الجمود. أما في الربط القياسي فيعني وضع الأعمال في جدول مرتبة حسب تسلسل أرقام مصادر الحاسوب التي تدل على أولويات الربط.

3- التحقق من حدوث حالة العناق الميت وحل تلك الحالة لحظة حدوثها Detect and Recover

وتعني هذه الطريقة ترك العمليات تحت التنفيذ, ومراقبة حالة الجمود, حتى إذا وقعت تم معالجة تلك الحالة في حينه ويلزم لذلك جدولان هما:

أ- جدول تعيين المصادر(Resource assignment table):

وظيفة هذا الجدول تحديد رقم المصدر ورقم العملية المرتبطة بذلك المصدر

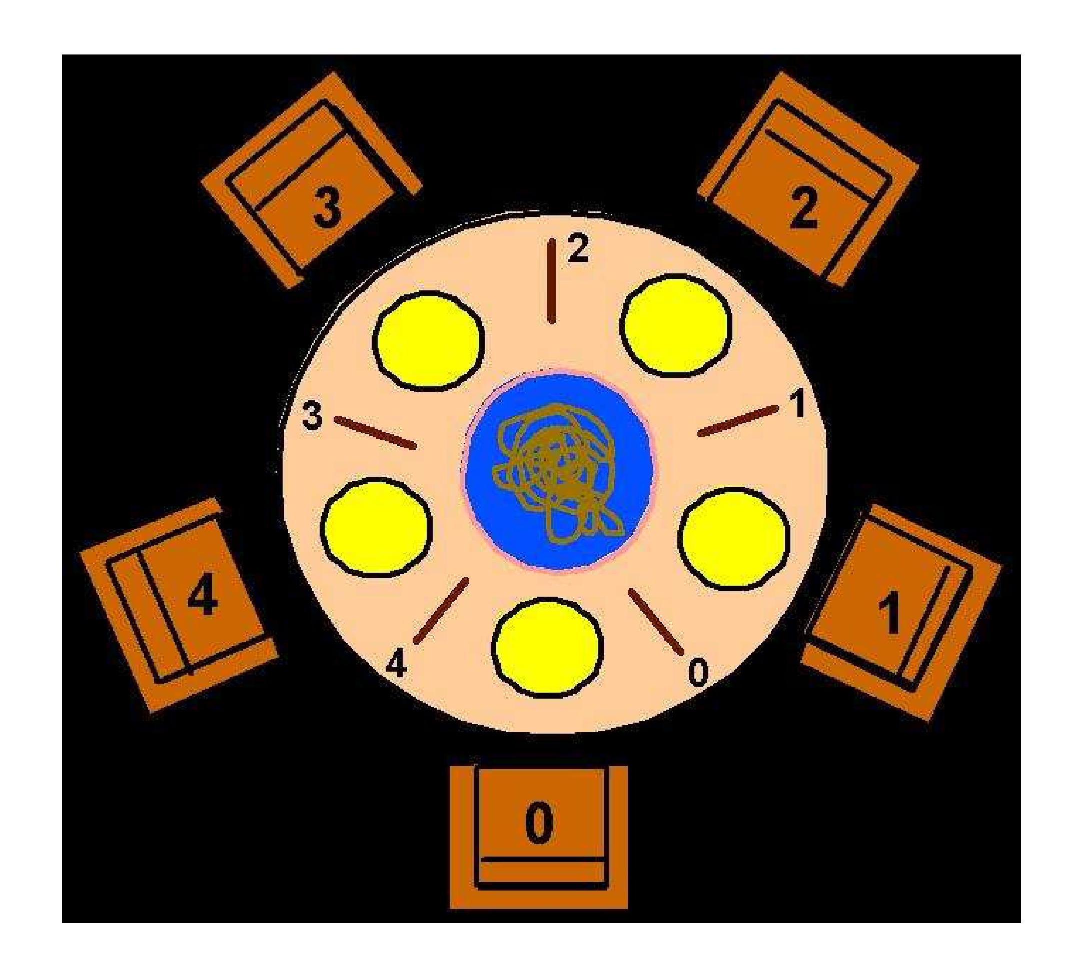
ب- جدول العمليات المنتظرة(process wait table):

وظيفة هذا الجدول تحديد أرقام العمليات الواقعة في حالة الانتظار وأرقام المصادر المرتبطة بتلك العمليات المنتظرة

يتعاون الاثنان على تحديد حالة الجمود وحلها.

***طعام الفلاسفة:**

وقد مثل على مشكلة الجمود **ادغار ديكسترا** في 1968 بأمر مشهور وهو مشكلة طعام الفلاسفة(**Dining Philosophers**):



فلو كان هناك خمسة فلاسفة يجلسون حول طاولة مستديرة وكل فيلسوف يقضي وقته مابين التفكير والأكل وفي وسط الطاولة صحن كبير من السباغيتي وسيحتاج الفيلسوف إلى شوكتين ليتناول حصته من السباغيتي..ولسبب ما ..لن يأخذ الفلاسفة سوى خمسة شوك طعام ولذا ستوضع شوكة واحدة بين كل زوج من الفلاسفة وقد اتفقوا على أن كل منهما سيستخدمها عند حاجته لها بيده اليمين أو الشمال حسب موقع الشوكة منه

وفي هذا الموقع (http://www.doc.ic.ac.uk/~jnm/book/book_applets/Diners.html)

ستجدين برنامج يوضح الوقت الذي يمضيه الفيلسوف في الأكل أو التفكير. لاحظي أن:

اللون الأصفر يعني أن الفيلسوف يفكر.

اللون الأزرق يعني أن الفيلسوف جائع.

اللون الأخضر يعني أن الفيلسوف يأكل.

استخدمي زر التجميد"Freeze" وزر المتابعة"Restart" للتحكم بالمحاكاة. وجربي تحريك الشريط إلى أقصى اليسار ولا حظي ماذا سيحدث؟؟

بعد بضع ثوان ، سيصبح كل من الفلاسفة جائعاً وسيمسك كل منهم شوكة واحدة معه وسينتظر توفر شوكة ثانية له وبما أنه لا يوجد المزيد من الشوك وكل منهم يحتاج لشوكتي طعام حتى يتمكن من الأكل فإن أياً منهم لن يتمكن من الأكل وسيصلون لمأزق الجمود وعدم القدرة على المتابعة.

يقضي الفلاسفة حياتهم بين التناوب على التفكير والأكل ، الفيلسوف بحاجة لالتقاط الشوك التي عن يمينه وعن شماله ليقوم بعملية الأكل، عندما يبدأ الفيلسوف بالتفكير ، يضع الشوك على الطاولة مرة أخرى.

وخلال عملية الأكل يكون أحد الفلاسفة يفكر أو جوعان أو...

وسنلاحظ أن مشكلة طعام الفلاسفة توفرت فيها شروط حدوث الجمود.

-فيلسوف واحد فقط يستطيع استخدام الشوكة في الوقت الواحد، ما يعني أن الشوكة مورد مقصور على فيلسوف واحد فقط.

-الفيلسوف الجائع والذي معه شوكة واحدة فقط سوف يحتفظ بها ويحتجزها وسينتظر الحصول على شوكة أخرى.

-وبما أن الفلاسفة مسالمون ومهذبون فلا أحد منهم سيرغب أو يحاول انتزاع الشوكة من جاره بالإجبار.

اتبعي **الرابط السابق** لرؤية البرنامج الآخر الذي حلَ المشكلة السابقة. حيث تجنب احتمال حدوث الجمود بأن جعل عدداً زوجياً من الفلاسفة يأخذون الشوك من البقية بترتيب مختلف، وهو أول اليسار

بدلا من أول اليمين. وهذا الحل يختلف عن الحلول الثلاثة المقترحة السابقة فهو لم يلغِ أياً من الشروط الثلاثة وإنما فرض ترتيباً معيناً لحجز الموارد. الأمر الذي يعني أنه من **المستحيل أن يمسك كل من الفلاسفة الخمسة بشوكة واحدة فقط.**

البرامج موجودة على

- http://www.doc.ic.ac.uk/~jnm/book/book_applets/Diners.html
- http://www.doc.ic.ac.uk/~jnm/book/book_applets/FixedDiners.html

هذا الفصل من إعداد ما يلي:

الموضوع	إعداد	المصادر
حلول الجمود	رزان المزروع	اختصارا من كتاب أنظمة التشغيل للدكتور زياد القاضي
طعام الفلاسفة	إيمان الريس ولمياء السدحان	http://courses.cs.vt.edu/csonline/OS/Lessons/Deadlock/index.html http://en.wikipedia.org/wiki/Dining_philosophers_problem
تمثيل الجمود	رنا نايف العصيمي التميمي وأمل الحماد	Operating system concepts Book by Silberchatz Galvin Gagne http://www.math-cs.gordon.edu/courses/cs322/lectures/deadlock.html
تعريف الجمود	ابتهال باعظيم	http://www.cs.jhu.edu/~yairamir/cs418/os4/sld003.htm

شفاء الجمود	نهى الطياش	Operating system concepts Book by Silberchatz Galvin Gagne
الموارد	أمل الحماد	http://www.math-cs.gordon.edu/courses/cs322/lectures/deadlock.html
طرق التعامل مع الجمود	منيرة عبد الله بن هريس وأمل الحماد	http://www.cs.rpi.edu/academics/courses/fall04/os/c10/index.html http://www.math-cs.gordon.edu/courses/cs322/lectures/deadlock.html
أمثله على حدوث الجمود	رهام حافظ وأمل الحماد ومنى الماجد وابتهال باعظيم	http://www.cs.rpi.edu/academics/courses/fall04/os/c10/index.html http://en.wikipedia.org/wiki/Deadlock http://www.math-cs.gordon.edu/courses/cs322/lectures/deadlock.html http://www.cs.jhu.edu/~yairamir/cs418/os4/sld003.htm
سبب حدوث الجمود	أمل الحماد	http://www.math-cs.gordon.edu/courses/cs322/lectures/deadlock.html
شروط حدوث الجمود	أمل الحماد ومنى الماجد وابتهال باعظيم وإيمان الريس	http://en.wikipedia.org/wiki/Deadlock http://www.math-cs.gordon.edu/courses/cs322/lectures/deadlock.html http://www.cs.jhu.edu/~yairamir/cs418/os4/sld003.htm http://courses.cs.vt.edu/csonline/OS/Lessons/Deadlock/index.html

الفصل السابع:
إدارة الذاكرة والذاكرة التخيلية

إدارة الذاكرة والذاكرة التخيلية
Memory Management and Virtual Memory

مقدمة

تعتبر ذاكرة الوصول العشوائي (RAM) أحد أكثر أجزاء الحاسب تأثيراً على أداءه، إذ أن التقنيات التي يستعملها الجهاز للتعامل مع الذاكرة يمكنها أن تؤثر في أداء الجهاز بشكل كبير، لذلك كان هدف كثير من البحوث هو الوصول إلى تقنيات وخوارزميّات تمكننا من استعمال الذاكرة بأفضل شكل ممكن لضمان فعاليّتها.

نعرف أن الذاكرة تستعمل لتخزين الأوامر والبيانات التي تخص البرامج التي يعمل عليها المعالج في الوقت الحاضر (processes) ، وحيث أن هدفنا هو أن يعمل المعالج على أكبر عدد ممكن من البرامج دون حصول انحدار كبير في مستوى الأداء، فإن هدفنا أيضاً هو أن نفعّل استخدام الذاكرة بحيث تستوعب أكبر عدد ممكن من البرامج.

عند فحص سلوك المعالج يلاحَظ في كثير من الحالات أن المعالج لا يشترط أن يكون كامل البرنامج موجود في الذاكرة العشوائية ليتمكن من معالجته، مثل البرامج التي تحتوي على أقسامٍ الغرض منها معالجة حالات خاصة نادرة الحدوث فقط، وإن كان المعالج سيحتاج جميع أقسام البرنامج لمعالجته، فإنه غالباً لا يحتاج هذه الأقسام في نفس الوقت، بل في أوقات مختلفة خلال عملية المعالجة، وحتى نتجنب حجز مكان ثمين في الذاكرة لأجزاء قد لا تستحقها، تم الاستعانة بتقنية الذاكرة الافتراضية، التي تمكّن النظام من تنفيذ برامج لا تقع بالكامل في الذاكرة العشوائية بل الأجزاء الضرورية منها، إن تنفيذ الذاكرة الافتراضية ليس سهلاً، إذ أن التساهل في تطبيقها قد يقلل من أداء الجهاز بشكل كبير، في هذا الفصل سنناقش الذاكرة الافتراضية متمثّلة بطريقة طلب الصفحات. 52

جمع وإعداد وتنقيح أفنان السبيهين (afnan.s@gmail.com)

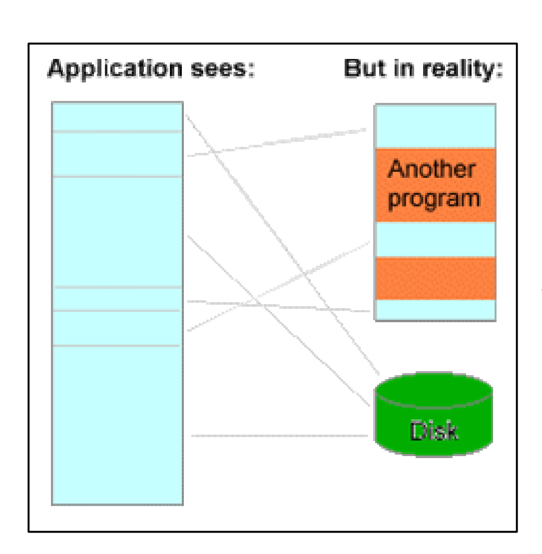
---

1 Operating System Concepts by Silberschatz, Galin and Gange.

أهداف الفصل

أولاً: خلفية

إن مبدأ تشغيل البرامج بدون تحميلها كاملةً إلى الذاكرة العشوائية يحمل في طيّه الكثير من الفوائد سواء على مستوى النظام أم على مستوى المستخدم، ففي هذه الحالة يمكن للمبرمجين برمجة العديد من البرامج الطويلة دون الأخذ بعين الاعتبار محدودية الذاكرة العشوائية للأجهزة المستهدفة، بالتالي تسهيل عملية صنع البرامج، كما أن المستخدمون لأنظمة تطبّق مبادئ الذاكرة الافتراضية يمكنهم العمل على عدة برامج في نفس الوقت، وإن كانت الذاكرة العشوائية لا تتحمل حجم جميع هذه البرامج مجتمعة.



إن تقنية الذاكرة الافتراضية تطبق مبدأ الفصل بين مفهومين للمساحة التي تمثّل البرنامج (الأوامر والبيانات): مساحة البرنامج الافتراضية التي يراها المبرمج أو المستخدم (virtual address space)، ومساحته الفعلية (actual address space) التي يطبّقها الجهاز عند تحميل وتشغيل البرنامج، إذ أن المبرمج يبرمج برنامجه واضعاً بعين الاعتبار مساحة افتراضية كبيرة إلى حد ما لبرنامجه، بينما في الواقع عند تشغيل هذا البرنامج فإن مساحته الفعلية في الذاكرة لا تطابق المساحة الافتراضية بل تغايرها من نواحي عدّة، فالمساحة الافتراضية –كما يتضح من اسمها- تفترض أن البرنامج مخزّن في الذاكرة ابتداءً من عنوان افتراضي (logical address) كصفر مثلاً، ثم تمتدّ مساحة البرنامج بشكل مستمرّ ومتصل، لكنّ مبدأ تقسيم البرامج إلى صفحات (pages) في فصل الذاكرة الرئيسية، يدلّنا على أن المساحة الواقعية للبرنامج مقسّمة إلى صفحات كما أن تخزين هذه الصفحات ليس متصلاً بل بحسب ما يتوفر لنا من أماكن شاغرة في الذاكرة الرئيسية، كما أن عنوان الصفحات الافتراضي سيتم ترجمته إلى عنوان حقيقي من قبل وحدة إدارة الذاكرة (Memory Management Unit).

(أفنان السبيهين)

وبذلك نستنتج أن هذا النوع من الذاكرة يعطي صورة للمستخدم بأن البرنامج تم تحميله بالكامل على الذاكرة الرئيسية خلال فترة المعالجة، و تعطي البرنامج التطبيقي الانطباع بأن هناك ذاكرة متصلة يعمل عليها، بينما في الحقيقة البرنامج مجزّأ إلى أجزاء وعند تحميل البرنامج لمعالجته يتم تحميل بعض هذه الأجزاء

للذاكرة والبعض الآخر يتم وضعه في مساحة في القرص الصلب (مساحة داعمة – backing store).

الذاكرة التخيلية ليست عبارة عن استخدام مساحات في القرص الصلب لتوسعة مساحة الذاكرة الحقيقة فقط, ولكنها تهدف إلى مخادعة البرامج لتظن أنها تستخدم مساحات كبيرة ومتجاورة من الذاكرة استعانةً بالقرص الصلب، بالتالي فإن القرص الصلب يوصف بأنه في هذه الحالة يتظاهر بأنه جزء من الذاكرة العشوائية، ويوهم البرامج بأنه امتدادٌ لها.[53]

(سارة الشثري)

يقوم نظام التشغيل بالبحث عن الأجزاء الغير مستعملة باستمرار من الذاكرة العشوائية ثم يقوم بنسخها في القرص الصلب وبذلك فانه يتيح مساحه أكبر في الذاكرة العشوائية حتى يتمكن نظام التشغيل من استخدام جزءاً أكبر من الذاكرة العشوائية في تشغيل تطبيقاته.

عملية التبديل التي تتم بين الذاكرة العشوائية و القرص الصلب تتم في شفافية تامة فمستخدم الجهاز يكاد أن لا يشعر بهذه العملية، وبذلك نكون قد حظينا بجزء أكبر من الذاكرة العشوائية وأيضا زيادة في عدد التطبيقات التي يعمل عليها المستخدم في الجهاز.

(نوال باعبد الله)

هذه التقنية تعطي عدة تسهيلات، فالنظام الذي يطبق هذا المبدأ يعمل بشكل جيد مع بيئة البرامج المتعددة، كما أنه يتم تقليل وقت الانتظار بشكل كبير في ظل وجود هذه الذاكرة، كما أن من مميزات تقنية الذاكرة الافتراضية أن حجم البرنامج لا يؤثر كثيراً في عدد البرامج التي يمكن تحميلها للذاكرة، بالتالي إعطاء لا محدودية في عدد البرامج المتزامنة، مما يعطينا استخداماً أكثر كفاءة للذاكرة، و لا يمكننا تجاهل ما تقدمه هذه التقنية من تسهيلٍ لعملية مشاركة الأوامر أو البيانات بين البرامج التي تقع في الذاكرة.

بعدد ما تقدمه هذه التقنية من فوائد ومميزات فإنها تحتوي عدداً من العيوب، فهذه التقنية قد تكون مكلفة من نواحٍ عدة، إما من ناحية المساحة الداعمة (backing store) أو من ناحية الوقت، فهي تزيد بشكل لا يمكن تجاهله عدد مرات مقاطعة البرامج، كما أنها من الممكن أن تزيد تعقيد مهمة البرمجة.[54]

---

[1] en.Wikipedia.org

1 Microcomputer Operating Systems book, Operating System Concept book.

(أمينة العبيد)

ثانياً: طلب الصفحات

عند القيام باستخدام برنامج ما فان هذا البرنامج يتم تقسيمه لصفحات Paging وهذه الصفحات Pages يتم تحميلها للذاكرة لكي يتم تنفيذ هذا البرنامج، في آلية طلب الصفحات في الذاكرة التخيلية Demand Paging in Virtual Memory لا يتم تحميل جميع صفحات البرنامج المشغل إلى الذاكرة الفعلية، إنما فقط الصفحات المطلوبة لتنفيذ العمليات التي يقوم بها المستخدم لهذا البرنامج، وهذه الآلية تعتبر أفضل بكثير من تحميل كامل البرنامج إلى الذاكرة (جميع صفحات البرنامج)، وذلك لأن أجزاء من البرنامج قد لا تستخدم خلال تشغيله، مما يعني حجز جزء من الذاكرة لصفحات من البرنامج غير مستخدمه، هذه الآلية تمكننا من تحميل عدة برامج كبيرة في آن واحد نظرا لأن كل برنامج لن يتم تحميله بالكامل، مما يوفر مساحة في الذاكرة لبرامج أخرى.

وحين يريد المستخدم استخدام جزئية جديدة من البرنامج موجودة في صفحات لم يتم تحميلها إلى الذاكرة، سيتم تحميل الصفحات المطلوبة عن طريق المبدل الكسول Lazy Swapper أو بالأصح Pager حيث إن لفظ Swapper يستخدم حينما يتم التعامل مع كل البرنامج دون تقسيمه إلى صفحات، ولفظ Pager يستخدم حين يتم التعامل مع الصفحات Pages كل على حدة.

و حينما يحاول برنامج ما الدخول على صفحة غير موجودة في الذاكرة الفعلية سينتج عن ذلك ما يعرف بخطأ الصفحة (Page-fault) مما يجعل الــ Pager يجلب الصفحات المطلوبة من القرص الصلب إلى الذاكرة لكي يتم استخدامها.

عند تطبيق هذه الآلية يمكننا أن نبدأ باستخدام برنامجٍ ما دون أن تكون أي من صفحاته قد تم تحميلها للذاكرة الفعلية، وخلال تنفيذ البرنامج يتم تحميل ما يحتاجه المستخدم من صفحات للذاكرة لتنفيذ العمليات المطلوبة.[55]

(نورة سلمان بن سعيد – حليمة حكمي)

مبادئ رئيسية

[55] En.wikipedia.org and www.science.unitn.it

نستخلص مما سبق أن المبدأ الأساسي لتقنية طلب الصفحات هو عدم تحميل جميع صفحات البرنامج من الذاكرة الثانوية (Secondary Storage) أو القرص إلى الذاكرة الحقيقية (Physical Memory)مباشرة عند بداية تنفيذ البرنامج، بل تحميلها حسب الحاجة **أثناء** التنفيذ.

عندما يتم تحميل (تبديل) البرنامج إلى الذاكرة الحقيقية للمرة الأولى فإن المصفح يحاول التنبّؤ بالصفحات التي سيتم استخدامها قبل إرجاعه إلى الذاكرة الثانوية مرة أخرى، هذه الصفحات هي التي سيتم إحضارها إلى الذاكرة الحقيقية بدلاً من إحضار كامل البرنامج، وبذلك تم اختصار الوقت الذي كان سيقضيه لو أنه اضطر إلى جلب عدد أكبر من الصفحات، فضلاً عن اختصار المساحة التي كانت ستملؤها تلك الصفحات.

هنا تظهر لنا الحاجة لمعرفة كافّة الصفحات التي تخص البرنامج: أيّها في الذاكرة الحقيقية وأيّها في الذاكرة الثانوية (لم يتم جلبها بعد)، ولحل هذه المشكلة نحتاج إلى دعم الجهاز لإضافة bit إضافي في الجدول الذي يربط بين كل صفحة و مكانها في الذاكرة الحقيقية (Page Table). هذه الخانة الإضافية تعطي إحدى قيمتين:

1- صحيح (valid) : تعني أن هذه الصفحة موجودة حاليّاً في الذاكرة الحقيقية.
2- غير صحيح (invalid) : تحتمل إحدى معنيين:
   أ. الصفحة خاطئة (لا يحتويها البرنامج أساساً).
   ب. الصفحة موجودة لكن لم يتم تحميلها إلى الذاكرة الحقيقية بعد. (لا تزال في القرص)

ويعرف النظام ما إذا كانت الصفحة المطلوبة موجودة على القرص (لم تجلب بعد) بإحدى طريقتين:

1- مكان الصفحة في الجدول محدد بعلامة (invalid).
2- مكان الصفحة في الجدول مربوط بعنوان لا يوجد في الذاكرة الحقيقية بل يوجد في القرص.

خطأ الصفحة

ذكرنا أنه عند بداية تنفيذ البرنامج للمرة الأولى فإن المُصفّح (pager) يحاول التنبؤ بالصفحات التي سيتم استخدامها، المُصفح يحاول جاهداً أن يكون هذا التنبؤ صحيحاً، إذ لن يحتاج إلى جلب صفحات إضافية من القرص أثناء التنفيذ. لكن لو أن النظام أثناء التنفيذ أراد صفحة لم تكن ضمن الصفحات التي تنبأ بها،

أي أنها لا تزال في القرص، فإنه عند الاستعانة بالجدول سيجد أن هذه الصفحة "غير صحيحة" بالتالي سيتسبب في حصول تعثّر (Trap) خاص يدعى **بخطأ الصفحة** (Page-Fault Trap) ، هذا التعثر حاصل نتيجة تقصير النظام في إحضار صفحات البرنامج إلى الذاكرة.

حينها سيضطر النظام للقيام بعدة خطوات للتعامل مع هذا التعثر: سيضطر المصفح إلى إحضار هذه الصفحة من القرص إلى الذاكرة،ثم تسكينها في مكان (Frame) خالٍ في الذاكرة، ثم سيعدل الجدول ليعكس وجود هذه الصفحة، ثم أخيراً يتم إعادة العملية التي طلبت هذه الصفحة.

نأخذ في الاعتبار حالة خاصة، وهي أن النظام لن يجلب أي صفحة تخص البرنامج عند بداية تنفيذه، أي أن كل أمر يتم تنفيذه في هذا البرنامج سيتسبب في خطأ الصفحة (Page-Fault) ، إلى أن يتم جلب جميع الصفحات التي يحتاجها ثم سيكمل تنفيذه بشكل طبيعي، تدعى هذه الحالة **بطلب الصفحات النقي** (Pure Demand Paging).[56]

(أفنان السبيهين)

أنواع خطأ الصفحة

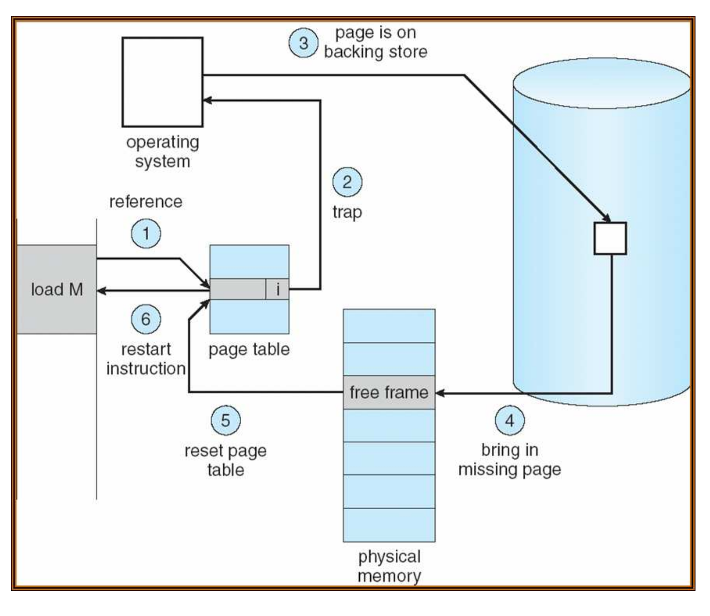
- **خطأ الصفحة الطفيف**: وهو أن تكون الصفحة المطلوبة موجودة في الذاكرة ولكن لم يتم تحديث الجدول ليعكس وجودها، وتحدث هذه الحالة إذا كان هناك ذاكرة مشتركة بين برنامجين أو أكثر فتكون الصفحة المطلوبة جاءت إلى الذاكرة عن طريق البرنامج الآخر.

- **خطأ الصفحة العظيم**: وهو أن تكون الصفحة المطلوبة غير محملة في الذاكرة وهذا الخطأ مكلف لأننا نأخذ بالاعتبار الوقت الذي يستغرقه جلب الصفحة من القرص.

- **خطأ الصفحة الباطل**: ويحدث هذا الخطأ عند محاولة القراءة من مكان فارغ في الذاكرة.

- **خطأ الحماية**: ويحدث هذا الخطأ عندما لا يستطيع البرنامج القراءة أو الكتابة في الصفحة المطلوبة ويحل نظام التشغيل مشكلة الكتابة بواسطة آلية النسخ عند الكتابة (copy-on-write)، وهو مشاركة نفس النسخة بين أكثر من برنامج ,وعند ما يريد أحد البرامج التحديث يعمل نظام التشغيل نسخة خاصة به .[57]

[56] Operating System Concepts by Silberschatz, Galin and Gange.
[57] En.wikipedia.org



خطوات التعامل مع خطأ الصفحة:

أولاً: يتم التأكد من الجدول الداخلي (عادة يكون محفوظ مع الـــPCB) للعملية ؛ ليتم تحديد ما إذا كانت الصفحة المطلوبة صحيحة (valid) أم غير صحيحة (invalid)، اعتماداً على ما إذا كانت تم تحميلها للذاكرة أم لا.

ثانياً: إذا كانت الصفحة غير صحيحة لكونها محمية من ذلك البرنامج (لا تنتمي له) إذن سيتم إنهاء العملية، أما إذا كانت الصفحة صحيحة لكن غير موجودة في الذاكرة إذن يتم إحضارها.

ثالثاً: يتم البحث عن إطار (frame) فارغ، لتوضع به الصفحة التي سيتم جلبها، وذلك بأخذه من قائمة الإطارات الفارغة (free-frame).

رابعاً: تتم جدولة عمليات القرص ليقرأ الصفحة المطلوبة إلى الإطار الجديد.

خامساً: عندما ينهي القرص عملية القراءة، يقوم النظام بتعديل الجدول الداخلي للعملية و جدول الصفحة، دليلاً على وجود الصفحة في الذاكرة.

سادساً: يتم إعادة تنفيذ الأمر الذي قُطع بتعثر نظام التشغيل.

الآن العملية تستطيع الوصول إلى الصفحة بما أن الصفحة جلبت إلى الذاكرة.[58]

(ليلى البيشي – نهلة محمد)

أداء تقنية طلب الصفحات

ثالثاً: النسخ عند الكتابة Copy-on-Write

نعرف من فصول سابقة أنه عندما يستخدم برنامج ما الـــ Fork() فإنه ينتج برنامجاً آخراً، هذا البرنامج الجديد (الابن) يكون نسخة طبق الأصل من البرنامج الذي أنتجه (الأب)، أي أن صفحات الأب يتم

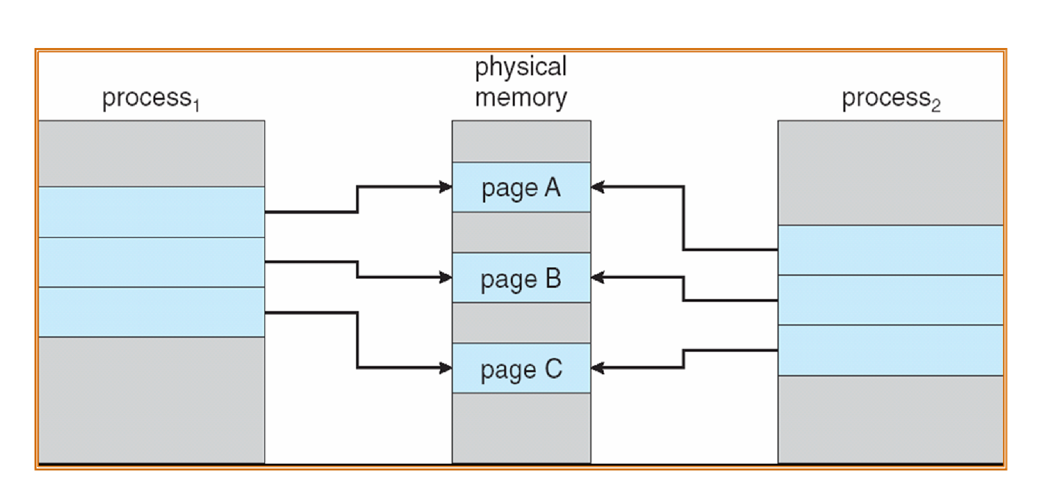
[58] Operating System Concepts by Silberschatz, Galin and Gange

نسخها كما هي لتكون صفحات الابن، يتضح لنا ما في هذه العملية من إهدار للمصادر إذ أن الابن نادراً ما يُترك ليكون نسخة عن أبيه بل سيستخدم الـــ exec() فلا تكون له نفس صفحات أبيه.

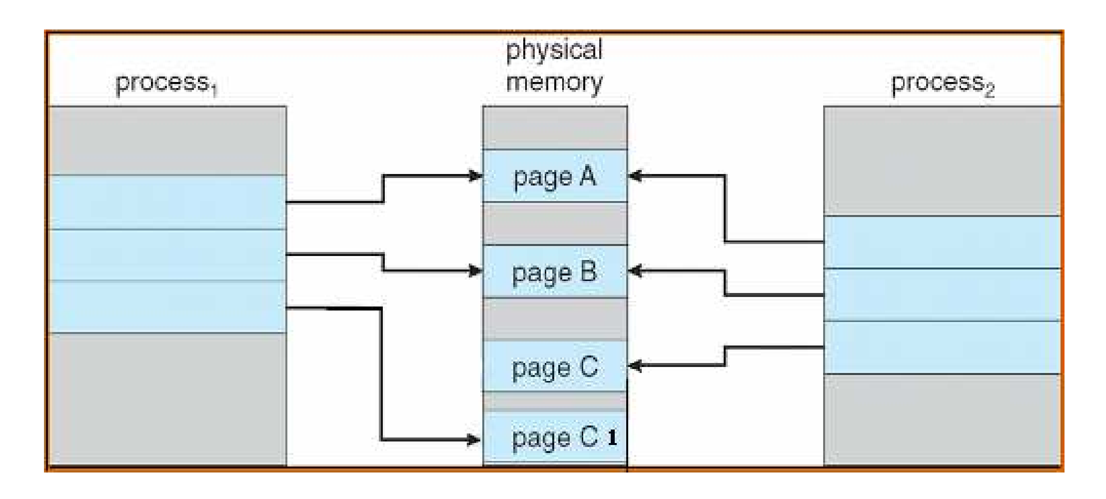
تقنية النسخ عند الكتابة (COW)، تسمح لبرنامجين الاشتراك في نفس الصفحات دون أن تسند نسخة لكل برنامج، إلى إن يحاول أحد هذين البرنامجين التعديل على إحدى الصفحات المشتركة، حينها يتم نسخ تلك الصفحة له، ليعدّل عليها كيفما شاء.

مثال:

لنفرض أن كل من (process1 - child) و (process2- parent) يؤشران على الصفحات Aو B و C



عند محاولة (process1) التعديل على الصفحة (C) ، ننتج نسخة جديدة من هذه الصفحة لتعكس هذه التعديلات، ونجعلها ملكاً لـــ(process1)، فيؤشر عليها.



(خلود الرومي)

رابعاً: استبدال الصفحات

علمنا أنه عند تطبيق تقنية طلب الصفحات، قد تحتاج أحد البرامج التي يعمل عليها المعالج حاليّاً إلى صفحة لم يتم تحميلها إلى الذاكرة، مما ينتج لنا تعثراً أسميناه بخطأ الصفحة، ونعرف أيضاً أن خطأ الصفحة يمكن معالجته بأن يتم إحضار هذه الصفحة من القرص إلى الذاكرة، لا يمكننا تجاهل المشكلة الأساسية التي قد يواجهها النظام عند هذه النقطة، ألا وهي عدم وجود مكان فارغ في الذاكرة، لذلك يطبق النظام تقنيّات على أساسها يختار صفحة (الضحيّة) ليستبدلها بالصفحة المطلوبة.

(أفنان السبيهين)

1-أساس استبدال الصفحات

إذاً فخوارزميات تبديل الصفحات هي التي تختار وتقرر الصفحة التي تخرج من الذاكرة عندما لا يكون هناك إطاراً (frame) فارغاً في الذاكرة، وتوضع الصفحة الخارجة (victim) في القرص (back store)، وتهدف هذه الخوارزميات إلى تقليل نسبة حدوث خطأ الصفحة، فكلما زادت عدد الإطارات بالذاكرة كلما قلت نسبة خطأ الصفحة.

يوجد في الحقيقة عدة خوارزميّات للتعامل مع هذه الحالة، الاختلاف الوحيد بين هذه الخوارزميات هي طريقة اختيار الضحية، وبينما تختلف فيما بينها في هذه الخطوة إلا أنها تتفق في بقية الخطوات المتبعة للتعامل مع الاستبدال هذه الخطوات تعرف بأساس استبدال الصفحات (basic replacement).

(حليمة حكمي)

هذه الخطوات هي:

1- البحث عن موقع الصفحة المطلوب جلبها من القرص.
2- البحث عن إطار فارغ في الذاكرة:
   a. إذا وجد إطار فارغ في الذاكرة، يتم استخدامه.
   b. إذا لم يجد إطاراً فارغاً يتبّع إحدى خوارزميات تبديل الصفحات لاختيار الإطار الضحية.
   c. يتم نقل محتويات الإطار الضحية إلى القرص، ويتم تعديل جدول الصفحات لتعكس عدم وجود هذه الصفحة في الذاكرة.
3- يتم إحضار الصفحة المطلوبة إلى الإطار الذي تمّ تفريغه في الذاكرة.
4- إعادة تنفيذ الأمر الذي طلب هذه الصفحة.

(صفا البلاع)

نلاحظ أنه في حالة عدم وجود مكان فارغ في الذاكرة، فإن النظام سيضطرّ أن ينقل صفحتين (صفحة خارجة للقرص وصفحة داخلة للذاكرة)، بالتالي فإن معالجة خطأ الصفحة يستهلك ضعف الوقت في حال عدم وجود إطار خالٍ في الذاكرة.

لكن لو أخذنا بعين الاعتبار أن الصفحة المطلوبة يتم **نسخها** من القرص ثم وضعها في الذاكرة، أي أن القرص يحتوي نسخة منها، حينها فلن نضطر لإرجاعها للقرص لو تم اختيارها لاحقاً لتكون ضحية، إلا إذا كان قد تم تعديلها، بالتالي يجب نقلها للقرص لتعكس الصفحة المخزنة فيه هذه التعديلات، لتطبيق هذه الطريقة نضيف خانة في الصفحة ونسمّيها خانة التعديل (modify bit or dirty bit)، بحيث أنه إذا تم التعديل على هذه الصفحة خلال تنفيذ البرنامج فإن هذه الخانة تحمل القيمة 1، وإذا لم يتم التعديل فإن الخانة قيمتها صفر، وهكذا إذا اختيرت صفحة لتكون ضحية فإن النظام يختبر خانة التعديل، إذا كانت الصفحة غير معدّلة (قيمة الخانة صفر) فإنه لا يجب نقلها إلى القرص، بل يتم استبدالها بالصفحة المطلوبة مباشرة، أما إذا كانت الصفحة الضحية معدّلة (قيمة الخانة واحد) فإنه يتم نقلها للقرص قبل أن يتم استبدال إطارها بالصفحة المطلوبة، وبهذا استطعنا أن نقلل من الوقت المطلوب لحلّ خطأ الصفحة إلى النصف **لو** كانت الصفحة لم يتم التعديل عليها من قبل.

وبهذا لإكمال تطبيق تقنية طلب الصفحات، نحتاج إلى **خوارزمية إسناد الإطارات** (**frame-allocation algorithm**) لتحديد كميّة الإطارات المسندة إلى برنامج ما في الذاكرة بحيث أن لا نسند للبرنامج أكثر مما يحتاج (over allocation)، أو أقل مما يحتاج، كما نحتاج أيضاً إلى **خوارزمية لاستبدال الصفحات** (**page replacement algorithm**).

كما قلنا سابقاً يوجد العديد من خوارزميات استبدال الصفحات، كل خوارزمية لها طريقتها الخاصة في اختيار الضحية في حال وجود خطأ الصفحة، ويتم تقييم واختيار واحدة منها على أساس الأقل تسببا في خطأ الصفحة، نقوم باختبار كل خوارزمية بتطبيقها على سلسلة من طلبات الصفحات (reference string)، كل صفحة تُمثّل برقم، إذاً فهي سلسلة من الأرقام، ومن ثم نحسب عدد مرات حدوث خطأ الصفحة (رقم الصفحة المطلوبة غير موجود بالذاكرة) لكل خوارزمية.

(أفنان السبيهين)

2-خوارزمية الداخل أولاً يخرج أولاً FIFO

تعتبر هذه الخوارزمية من أبسط خوارزميات تبديل الصفحة، إذ أن هذه الخوارزمية تربط كل صفحة مع الوقت الذي أُحضرت فيه للذاكرة، وعندما نختار صفحة لنستبدلها فإننا نختار أقدم صفحة دخلت الذاكرة، كما أنه ليس بالضرورة أن نسجل وقت دخول الصفحة للذاكرة، بل يكفي أن ننشئ صفّاً (FIFO queue) يمثّل كل الصفحات بالذاكرة على ترتيب قدومها، وعند إحضار صفحة للذاكرة تكون في آخر الصف (لأنها صفحة حديثة)، بالتالي فإن الصفحة التي في مقدمة الصف هي أقدم صفحة.

(خديجة أخرفي – ليلى البيشي)

مثال:

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2	2	2	4	4	4	0	0	0	7	7	7
	0	0	0	3	3	3	2	2	2	1	1	1	0	0
		1	1	1	0	0	0	3	3	3	2	2	2	1

page frames

– أولاً نبدأ بثلاث إطارات فارغة في الذاكرة، سيحدث خطأ صفحة لأول ثلاث طلبات وهي 7 و 0 و 1، لكنه سيجد إطاراً فارغاً لكل واحدة منها فلا نضطر لاستخدام تقنيات استبدال الصفحات.

– عند طلب الصفحة رقم اثنين، لن يجد لها مكاناً فارغاً بالتالي سيبحث عن أقدم صفحة تم جلبها للذاكرة ليستبدلها، هذه الصفحة هي الصفحة رقم 7، بالتي ستحل صفحة 2 بدل صفحة رقم 7.

– طلب صفحة رقم صفر لن يسبب خطأ صفحة، لأن الصفحة صفر موجودة بالذاكرة أساساً.

– طلب الصفحة 3 يسبب خطأ صفحة لأنها غير موجودة بالذاكرة، ولأنه لا يوجد مكان فارغ لنجلبها له، سنضطر لتطبيق تقنية استبدال الصفحات لتحديد الضحية، الضحية في هذه الخوارزمية هي أقد صفحة تم جلبها للذاكرة، في هذه الحالة تحل الصفحة 3 بدلاً من الصفحة صفر.

– وهكذا نطبق هذه الخطوات لكل رقم في سلسلة الطلبات، في النهاية سنجد أنه عند تطبيق تقنية الداخل أولاً خارج أولاً لاستبدال الصفحات على هذه السلسلة، ينتج 15 خطأ صفحة.

(نورة الحماد)

نعرف بديهيّاً أن زيادة عدد الإطارات الفارغة في الذاكرة المعطى لكل برنامج، يمكن أن يقلل من احتمال العدد الكلي لأخطاء الصفحة التي تتسبب بها تقنية استبدال الصفحات، ولأن لكل قاعدة شذوذ، فإن الشذوذ عن هذه القاعدة يدعى **شذوذ بيلادي** (Belady's Anomaly).

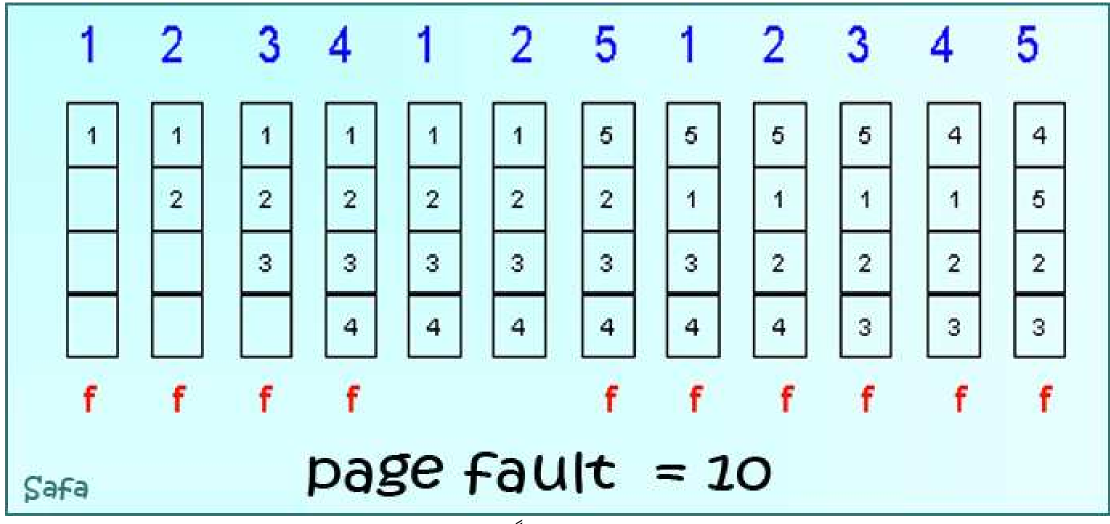
(أفنان السبيهين)

تعاني خوارزمية الداخل أولاً خارج أولاً من هذا الشذوذ، فإن زيادة عدد الإطارات المعطى لكل برنامج، في نظامٍ تطبق فيه خوارزمية الداخل أولاً خارج أولاً، لا يعني بالضرورة أن العدد الكلي لأخطاء الصفحات سيقل.

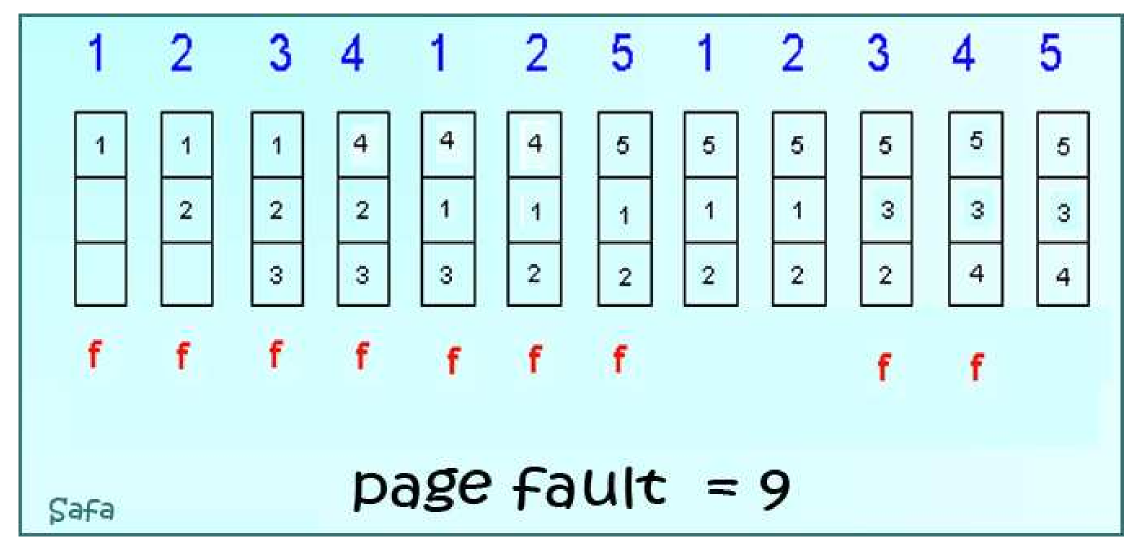
(خديجة أخرفي)

مثال:

نرى مثالاً يوضح شذوذ بيلادي، أولاً بتطبيق سلسلة الطلبات على أربعة إطارات ينتج لنا عشرة أخطاء (f).



والآن نطبق نفس السلسلة لكن على ثلاث إطارات بدلاً من أربعة، فينتج لنا 9 أخطاء فقط.



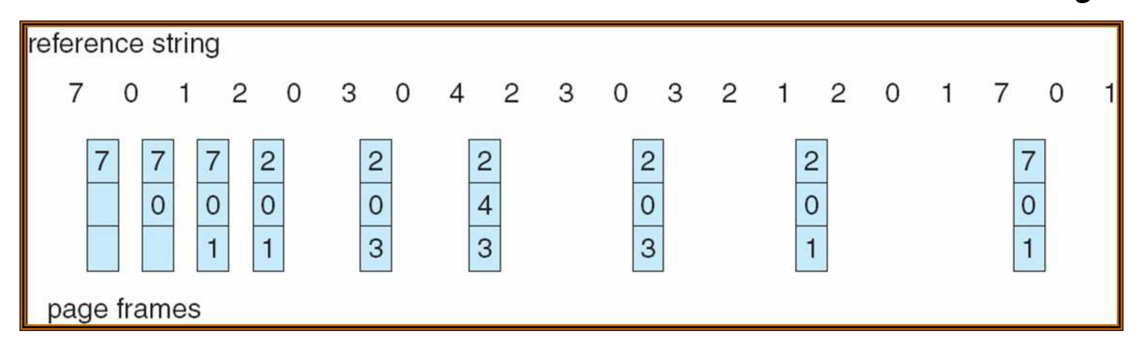
(صفا البلاع)

3- الخوارزمية المثاليّة

نتج الكشف عن شذوذ بيلادي أن أراد الباحثون الحصول على طريقة مثالية لاستبدال الصفحات، هذه الطريقة هي أقل الخوارزميات تسبباً في خطأ الصفحة كما أنها لا تعاني من شذوذ بيلادي، تقوم هذه الطريقة على اختيار الصفحة التي لن يتم استخدامها قريباً، أو سيتم استخدامها بعد فترة طويلة بالنسبة لبقية الصفحات الموجودة بالذاكرة، أي يجب أن نعرف بقية سلسلة الطلبات والذي لا يُعطى للنظام جاهزاً وإنما تحدث هذه الطلبات في أوقات زمنية متفرقة، يعني هذا أنه لنعرف أيّ الصفحات لن يتم طلبها لوقت طويل في المستقبل يجب أن نعرف ماذا سنحتاج في المستقبل، لهذا يستحيل تطبيق هذه الخوارزمية لأنها تتطلب معرفة بالمستقبل الذي لا يمكن التنبؤ به، الفائدة من هذه الخوارزمية هو استخدامها كمعيار مقارنة مع بقية الخوارزميات لمعرفة مدى فعاليتها، فكلما كانت الخوارزمية نتائجها قريبة من نتائج الخوارزمية المثاليّة كلما كانت أفضل.

(صفا البلاع – ليلى البيشي – خديجة أخرفي)

مثال:



– أولاً نبدأ بثلاث إطارات فارغة في الذاكرة، أول ثلاث طلبات ستتسبب في خطأ الصفحة، لكنها ستجد إطاراً فارغاً.

– ثانياً، يتم طلب الصفحة رقم 2، فتتسبب بخطأ الصفحة لعدم وجودها في الذاكرة، ولأنه لا يوجد إطار فارغ لتوضع فيه، سنستخدم تقنية استبدال الصفحات، بالخوارزمية المثالية وبالنظر في بقية سلسلة الطلبات، نجد أن الصفحة صفر سنحتاجها قريباً جداً، كما أننا سنحتاج الصفحة 1 قبل أن نحتاج الصفحة 7، فنستنتج أن الصفحة 7 هي التي لن نستخدمها قريباً مقارنةً بصفر وواحد، فنضع 2 بدلاً من 7.

– عند طلب الصفحة صفر لن تتسبب بخطأ صفحة لوجودها في الذاكرة.

– عند طلب الصفحة 3، سنحتاج لاختيار الصفحة الضحية، وهي الصفحة التي لن تستخدم لأطول فترة من الزمن بين الصفحات الموجودة بالذاكرة، وهي في هذه الحالة الصفحة رقم 1 لأننا سنحتاج كل من صفر و 2 قبل أن نحتاجها، فنستبدل 1 بـــ 3.

– وهكذا إلى نهاية سلسلة الطلبات نجد أن المجموع الكلّي لعدد مرات حدوث خطأ الصفحة هو 9 أخطاء.

(نورة الحماد)

ولأننا قلنا أن الخوارزمية المثالية تضمن حدوث خطأ الصفحة بأقل نسبة ممكنة، فإنه لا يمكن أن نطبق أي خوارزمية على سلسلة الطلبات هذه وباستخدام ثلاث إطارات، إلا بحدوث 9 أو أكثر من الأخطاء.

4- خوارزمية الأقل استخداماً مؤخراً

هي خوارزمية تحاكي الخوارزمية المثالية، فبينما خوارزمية الداخل أولاً خارج أولاً تأخذ بعين الاعتبار وقت إحضار الصفحة للذاكرة، والخوارزمية المثالية تأخذ بعين الاعتبار وقت الاستخدام القادم لكل من الصفحات الموجودة بالذاكرة، فإن خوارزمية الأقل استخداماً مؤخراً تستخدم الماضي القريب بدلاً عن المستقبل، فتستبدل الصفحة التي لم يتم استخدامها لأطول مدة، أي الأقل استخداماً.

لتطبيق هذه النظرية تربط الخوارزمية كل صفحة بوقت آخر استخدام لها، وبذلك تعتبر هذه الخوارزمية تمثّل الخوارزمية المثاليّة لكن بالنظر للماضي بدلاً من المستقبل، مما يجعلها معقولة التطبيق.

مثال:

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2	2	4	4	4	0	1	1	1
	0	0	0	0	0	0	3	3	3	0	0
		1	1	3	3	2	2	2	2	2	7

page frames

- أولاً: سيتسبب طلب كل من الصفحات 7 و صفر و 1 بخطأ صفحة، لكن سيتوفر لهذه الصفحات إطارات فارغة، فلن نستخدم تقنية استبدال الصفحات.

- ثانياً: عند طلب الصفحة 2، ولعدم وجود إطار فارغ لوضعها به سنضطر لاختيار إحدى صفحات الذاكرة لتكون الضحية، باستخدام طريقة الأقل استخداماً مؤخراً نجد أن الصفحة رقم 7 هي أقدم صفحة تم طلبها في سلسلة طلب الصفحات، فيتم استبدالها بــ2.

- ثالثاً: عند طلب الصفحة صفر، لن يتسبب ذلك في خطأ صفحة لوجودها في الذاكرة.

- رابعاً: عند طلب الصفحة 3، وبعد تطبيق خوارزمية الأقل استخداماً مؤخراً نجد أن الصفحة 3 يجب أن تأخذ مكان الصفحة 1، لأنها كل من صفحة صفر وصفحة 2 تم استخدامهما قبل استخدام الصفحة 1، فهي الأقل استخداماً.

- وهكذا لنجد أن عدد أخطاء الصفحة الناتج عن تطبيق هذه الخوارزمية هو اثنا عشر خطأً.

(نورة الحماد)

هذه الخوارزمية هي الأكثر استخداماً بين بقية الخوارزميات، نظراً لأدائها الجيّد، وإمكانية تطبيقها، ولتطبيق هذه النظرية يمكن استخدام طريقتين لمعرفة أي الصفحات الموجودة بالذاكرة هي الأحدث استخداماً وأيها الأقدم استخداماً، هذين الطريقتين هما:

- **طريقة العداد**: في هذه الطريقة نسند لكل صفحة عداداً يحمل عدد المرات التي طُلِبت فيه هذه الصفحة، بحيث أنه كلما استخدمنا هذه الصفحة كلما زادت قيمة العداد، فعندما يتم طلب هذه الصفحة، تُنسخ قيمة الساعة الحالية إلى العدّاد، فيعكس العداد وقت آخر استخدام، وعندما يحتاج النظام إلى اختيار صفحة لإخراجها من الذاكرة فإنه يختار الصفحة ذات العداد الأقل قيمة، فالعداد الأقل قيمة يعني أنها الأقدم استخداماً، تتميز هذه الطريقة بسهولة تطبيقها.

- طريقة الــstack: في هذه الطريقة، يتم حفظ أرقام الصفحات في شكل متسلسلة بحيث أن الصفحة الأقدم تكون في الأسفل، أما الصفحة الأحدث استخداماً فتكون في الأعلى، بحيث أنه

عندما نستخدم صفحة ما، فإننا نخرجها من هذه المتسلسلة ثم نضعها في الأعلى، وهكذا تكون الصفحة الأخيرة هي الأقدم استخداماً، للقيام بهذه العملية يحتاج النظام إلى تغيير ستّ مؤشرات للحفاظ على ترتيب المتسلسلة، لكنها تتميز بأن النظام لا يحتاج للقيام بعملية بحث عن الصفحة الأقدم استخدام لأنها حتماً ستكون في الأسفل، فهذه الطريقة سريعة لكن يصعب تطبيقها نسبيّاً.

(حليمة حكمي – منى حكمي – صفا البلاع)

5- الخوارزمية التقريبية لخوارزمية الأقل استخداماً مؤخراً

تضيف بعض أنظمة الحاسب لكل مدخل في جدول الصفحات (أي لكل صفحة) خانة إضافية تدعى بخانة الطلب (reference bit)، بحيث أنه عندما تُطلب هذه الصفحة فإن هذه الخانة تحمل القيمة 1 ، هذه الخانة تساعد النظام لاحقاً في التعرف على الصفحات التي تم طلبها، والصفحات التي لم تُستخدم مطلقاً، رغم أن النظام لا يستطيع أن يعرف ترتيب طلب الصفحات التي تحمل القيمة 1 ، إلا أن هذه الخانة يمكن أن تساعد كثيراً في عملية استبدال الصفحات، فالصفحة ذات القيمة 1 هي بالتأكيد صفحة مستخدمة ومرغوبة من قبل النظام، لذا تميل أغلب الخوارزميات إلى عدم التردد في استبدال الصفحة التي تحمل القيمة صفراً في خانة الطلب، سنستعرض الآن عدّة خوارزميات مبنية على إضافة خانة الطلب في الجدول الذي يمثّل الصفحات الموجودة بالذاكرة.

أ- خوارزمية الفرصة الثانية

هذه الخوارزمية تبدأ بتطبيق خوارزمية الداخل أولاً خارج أولاً، بالإضافة إلى خانة الطلب، في البداية تكون خانة الطلب لكل الصفحات صفراً، ويتم تعديلها إلى 1 عندما يتم استخدام هذه الصفحة، وعندما يختار النظام صفحة لاستبدالها (بنظام الداخل أولاً خارج أولاً)، فإنه يفحص خانة الطلب، إن كانت مساوية لصفر تستبدل هذه الصفحة مباشرة، أما إن كانت تحمل القيمة 1 ، فإن النظام يعطيها فرصةً ثانية، أي لا يقوم بإخراجها من الذاكرة مباشرة، وعندما تنال إحدى الصفحات فرصةً ثانية فإن قيمة خانة الطلب التابعة لها تقلب إلى صفر، لنتأكد أنها لن تظل تنال فرصاً عديدة، حينها سينتقل النظام إلى الصفحة التي تليها في ترتيب الداخل أولاً خارج أولاً، وتُعامل هذه الصفحة بالمثل.

لتطبيق هذه الخوارزمية يتم ترتيب الصفحات على شكل حلقة، بحيث يبحث النظام في هذه الحلقة على الصفحة التي تحمل القيمة صفر في خانة الطلب، مغيراً في طريقة جميع الصفحات ذات القيمة 1 إلى

صفحات تحمل القيمة صفر، وهكذا فإن الصفحة التي تستخدم بشكل مستمر ستحافظ على القيمة 1 في خانة الطلب، أما الصفحات التي لم تستخدم منذ آخر تصفير (آخر فرصة أعطيت لها)، فإنها عند اختيارها مرة أخرى ستخرج من الذاكرة دون أن تعطى فرصة أخرى.

(صفا البلاع – أفنان السبيهين)

6- استبدال الصفحات المبني على العد

تبقي هذه الطريقة عداداً لكل صفحة، يحمل هذا العداد المرات التي تم طلب هذه الصفحة، بحيث أنه كلما طُلِبت صفحةٌ ما، فإن عدادها يزيد بواحد.

هناك خوارزميتين تستخدم هذه الطريقة لتختار الصفحة التي سيتم إخراجها من الذاكرة، اختلفت هاتين الطريقتين في دلالة هذا العداد، هل يجب إخراج هذه الصفحة لو كان عدادها أقل، أم لو كان عدادها أكبر؟

أ- خوارزمية الأقل استخداماً :

تستبدل هذه الخوارزمية الصفحة الأقل استخداماً، أي أن عدّادها يحمل الأقل قيمة، تستند هذه الخوارزمية على أن الصفحة ذات الاستخدام الأكثر، لها الحق في البقاء مدةً أطول في الذاكرة، لحاجة النظام لها، وتظهر لنا مشكلة الصفحة التي تستخدم بشكل مكثّف في بداية تنفيذ البرنامج، ثم لا تستخدم أبداً بعد ذلك، هذه الصفحة ستحمل الأعلى قيمة، لكنها في الحقيقة لا تستحق البقاء في الذاكرة.

ب- خوارزمية الأكثر استخداماً:

هذه الخوارزمية تعاكس الخوارزمية السابقة لحلّ المشكلة المطروحة، أي أنها تستبدل الصفحة الأكثر استخداماً.

يندر استخدام هاتين الطريقتين، لأن نتائجهما لا تمثّل نتائج الخوارزمية المثالية، كما أنهما تستهلكان جزءاً من المصادر لا يُستهان به.

(صفا البلاع – أفنان السبيهين)

الفصل الثامن:
نظام الملفات

11.1 : هيكلة نظام الملفات:

تستخدم المساحات الفارغة من الأقراص الصلبة (disk) لتوفير بيئة دائمة لتخزين الملفات في الذاكرة الثانوية (secondary storage) ؛ وذلك لتميزها بميزتين مهمتين :

1. قابلية هذه الأقراص لعمليات إعادة الكتابة :

حيث يمكن قراءة ملف معين و التعديل على هذا الملف ، ومن ثم تخزين الملف المعدل في نفس مكان الملف الأصلي.

2. إمكانية الوصول إلى أي معلومة مخزنة في القرص مباشرة بمعرفة مكان تخزينها.

هذا يعني سهولة الوصول إلى أي ملف إما باستخدام طريقة الوصول المتتالي ، أو باستخدام الوصول المباشر ؛ ما يعني أن عملية التبديل السريع بين الملفات تتطلب فقط تحريك رأس القراءة الموجود على سطح القرص إلى المكان المناسب والانتظار حتى تتم عملية نقل المعلومات.

لزيادة فعالية عمليات الإدخال والاخرج (I|O) ؛ فإنه بدلاً من نقل byte واحد فيك مرة ، يتم نقل البيانات بين الأقراص والذاكرة الرئيسية (RAM) على أساس وحدات تسمى blocks . كل block يحتوي على إقطاع [59] (sector) واحد أو أكثر ، على حسب نظام الملفات المستخدم.

لتوفير وصول ملائم و فعّال للقرص ، يقوم نظام التشغيل بدعم نظام ملفات واحد أو أكثر ؛ يتيح للمستخدم إمكانية حفظ الملفات ، واسترجاعها بسهولة.

نظم الملفات تطرح مشكلتين مختلفتين :

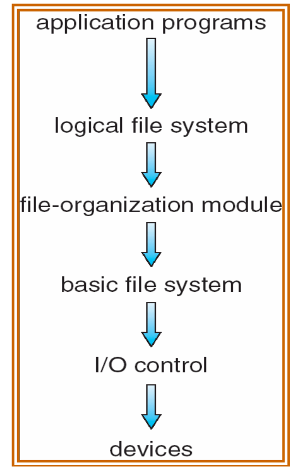
الأولى : كيف يجب أن يظهر نظام الملفات للمستخدم.

الثانية : تكوين الخوارزميات وهيكلة البيانات المناسبة لتحويل نظم الملفات الواقعية ، إلى أقراص التخزين الثانوية الفيزيائية.

يتكون نظام الملفات من عدد من المستويات ، كل مستوى يستخدم المميزات الخاصة بالمستوى الأقل منه ؛ ليقدم مميزات جديدة للمستويات الأعلى. (شكل 11.1)

---

[59] الإقطاع أو الـ sector هو أصغر جزء من القرص ، يختلف حجمه باختلاف القرص ، ويتراوح حجمه ما بين 32 بايت إلى 4,069 بايت = 4 ك.ب ، وغالباً ما يكون 512 بايت.



شكل 11.1: مستويات نظام الملفات

11.2 إنتاج نظام الملفات

11.2.1 نظرة عامة :

لتنفيذ نظام ملفات نستخدم مفهومين:

1-on-disk

2-in-memory

وهذان المفهومان يعتمدان على نظام التشغيل ونظام الملفات.

on-disk:

نجد أن نظام الملفات يحتوي معلومات عن كيفية إجراء عملية الـــ boot لنظام التشغيل المخزن هناك, ومجموع أعداد القطع (blocks) , وعدد الأماكن الخالية من القطع و أماكنها ,ودليل الهيكل .

وسنبدأ بوصف هذه الهياكل بإيجاز:

- boot control block: تحتوي على المعلومات التي نحتاجها في النظام لعمل إقلاع للنظام (boot). إذا كان القرص لا يحتوي على نظام تشغيل فإنه سيكون فارغ . وتسمى في UFS بـــ boot block. بينما في الـــNTFS تسمى partition boot sector .
- volume control block: يحتوي على تفاصيل الحجم مثل عدد الـــ blocks وحجمها وعدد الخالية منها ومؤشراتها وعدد FCB الخالية ومؤشراتها. في UFS تسمى بــــ superblock بينما في NTFS تسمى master file table

▪ الدايركتوري لكل ملف : يستخدم لتنظيم الملفات في UFS يحتوي الملف على الاسم والروابط بينما في NTFS يخزن في master file table .

▪ FCB يحتوي على تفاصيل عن الملف من تصريحات والمالك والحجم وموقع بيانات الـــblocks في UFS يسمى inode. في NTFS المعلومات تخزن ضمن master file table الذي يستخدم في قاعدة البيانات مع كل صف في الملف

in-memory:

المعلومات تستخدم لكلا من إدارة نظام التشغيل و تحسين الأداء.

وهذا التنظيم يحتوي على التالي :

1. in memory mount table : يحتوي على معلومات كل الأجزاء المتصلة.

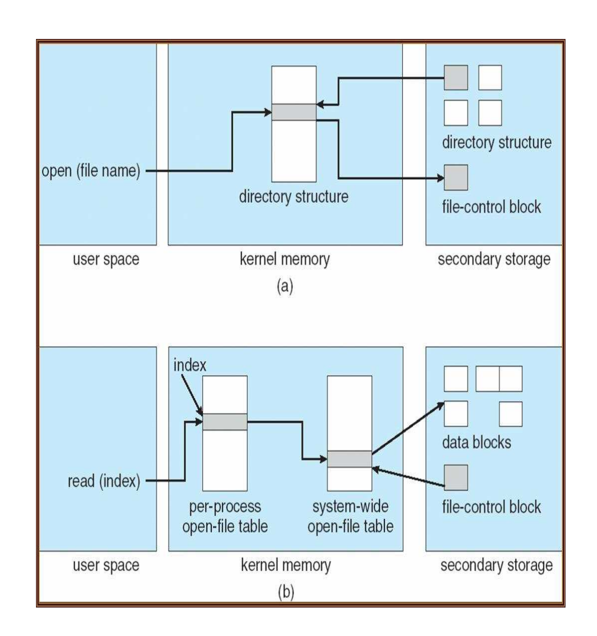
2. in memory directory structure cache : يخزن معلومات الأدلة المستخدمة مؤخراً.
3. system -wide open-file table يحتوي على نسخه منFCB لكل ملف مفتوح وكذلك المعلومات الأخرى.
4. per-process open-file table يحتوي على مؤشرات ملائمة للدخول في-system wide open-file table وكذلك المعلومات الأخرى.

الشكل التالي يوضح الــ FCB النموذجية :

file permissions
file dates (create, access, write)
file owner, group, ACL
file size
file data blocks or pointers to file data blocks

بعض نظم التشغيل مثل الــ UNIX تعالج الدليل بالضبط مثل الملف فقط باختلاف بسيط في خانة واحدة تشير إلى أنه دليل.

بينما نظم التشغيل الاخرى مثل WINDOWS NT يعامل الأدلة على أنها أجزاء مختلفة تماماً عن الملفات وذلك بتنيفذ أجزاء من النظام (system call) مختلفة لكل واحد منهما.



11.2.2 : **التجزيء والتركيب** : (Mounting and Partition)

تقسيم المحركات الصلبة :

التقسيم : هو عملية تجزيء إلكترونية منطقية للمحرك الصلب إلى مجموعات من الاسطوانات .

توفر الأقسام مرونة كبيرة في طريقة تنظيم هذه المحركات . مثلاً , ربما يحوي الحاسب محرك قرص صلب فيزيائياً وحيداً ,

و لكنه يتضمن من 1 إلى 24محركاً منطقياً أسمائهن من :C حتى :z .

تتيح لك عملية التقسيم تنظيم المحرك بحيث يناسب احتياجاتك الشخصية . مثلاً لقد قسمت المحرك الصلب لدي (GB30) إلى قسمين هما GB 25

للمحرك :C حيث قمت بتخزين windows 2000 وكل البرامج الأخرى وGB5 للمحرك D حيث قمت بتخزين المعلومات الخاصة بي .

كما يتيح التقسيم وضع أكثر من نظام على محرك القرص الصلب . إذ يمكن وضع نظام تشغيل على قسم ونظام تشغيل آخر على القرص الآخر .

تقسيم الأقراص الأساسية :

تولد عملية التقسيم عنصرين في محرك القرص الصلب وهما سجل الإقلاع الرئيسي MBR(Master Boot Record) وجدول التقسيم (Partition Table) . وعندما يقلع الحاسب من محرك القرص الصلب فإنه يبحث عن أول قطاع من المحرك الفيزيائي , والذي يدعى قطاع الإقلاع (boot sector) . يحتوي قطاع الإقلاع على سجل القطاع الرئيسي MBR وعلى جدول التقسيم . وإن MBR ليس أكثر من شيفرة صغيرة تأخذ التحكم بعملية الإقلاع من BIOS النظام . ويقوم MBR بمهمة وحيدة وهي البحث عن قسم ما في جدول التقسيم , والذي يحوي نظام تشغيل مقبول.
ويأخذ كل قسم يحوي نظام التشغيل على إعداداً خاصاً يدعى active (نشط) , والذي يستعمله MBR لتحديد أي نظام تشغيل سيقوم بتحميله . تدعم جميع جداول تقسيم محرك القرص الصلب حتى أربعة أقسام إقلاع , ودوماً يكون قسم واحد هو النشط . وهذا منطقي لأنك لا تستطيع تشغيل أكثر من نظام في نفس الوقت .
إن سجل الإقلاع الموجود في بداية محرك القرص الصلب هو ليس فقط قطاع الإقلاع الوحيد في المحرك . فأول قطاع من أول اسطوانة في كل قسم يحوي قطاع إقلاع يدعى وحدة تخزين قطاع الإقلاع (volume boot sector) . فبينما يعرف القطاع الرئيسي الأقسام , تخزن وحدات تخزين قطاع الإقلاع المعلومات الهامة لكل قسم , مثل موقع ملفات نظام التشغيل .

أنواع التقسيم :

يمكن أن يحتوي المحرك الصلب أربعة أقسام على الأكثر , سواء أكانت أقسام إقلاع أم لا . تصنف هذه الأقسام ضمن نوعين : أساسي (primary) وموسع Extended. حيث ينجز كل نوع مجموعة مختلفة من الوظائف . ويتم إنشاء هذه الأقسام وفق المتطلبات الخاصة للنظام .

الأقسام الأساسية :

تخزن أنظمة التشغيل في الأقسام الأساسية , وبالتالي إدا أردنا الإقلاع من محرك القرص الصلب فيجب أن يحتوي على قسم أساسي . يقوم MBR بفحص جدول التقسيم باحثاً عن القسم الأساسي .
يمكن أن يحتوي المحرك الصلب حتى أربعة أقسام أساسية . لكن توفر الأنظمة windows 9x برنامج التقسيم المدمج الذي يدعى FDISK , و الذي يتيح إنشاء قسم أساسي واحد فقط في القرص . أ ن Microsoft لم تكن تريدك تثبيت أي أنظمة تشغيل أخرى .مع أن المحركات الصلبة تعتمد أربعة أقسام أساسية , إلا أننا لا نصادف ذلك أبداً في معظم أنظمة windows 9x .

تدعم أنظمة التشغيل الأخرى مثل Linux , windows 2000 وجود عدة أقسام أساسية على نفس المحرك.
وتستعمل عدة مصطلحات للإشارة إلى هذه الخاصية , ولكن أكثرها شيوعاً هو الإقلاع المزدوج أو الإقلاع المتعدد . مثلاً أنا لدي أربعة أقسام أساسية يحوي كل منها نظام تشغيل وهي : RedHat Linux,windows98,windowsXP , windows2000, .
.بمعنى آخر , لقد جزأت المحرك إلى أربعة أقسام يحوي كل منها نظام تشغيل مختلف وللقيام بذلك استخدمت أداة طرف ثالث وهي system Commander 7 وذلك لإعداد الأقسام . تحتوي الأنظمة windows 2000, Linuxعلى أدوات مشابهة , لكن يفضل استخدام system commander . وعندما يقلع الحاسب يأخذ system commander القيادة من MBR ويسأل وفق أي نظام تريد الإقلاع .
وعند الإقلاع وفق نظام تشغيل معين لا يمكن عندها رؤية الأقسام الأساسية الأخرى. فمثلاً , عند الإقلاع وفق windows 98 لا ترى إلا المحرك :C . أيضاً , عند الإقلاع وفق Linux فإنك ترى القسم الخاص به فقط . أما عند الإقلاع وفق windows 2000 فإنك ترى الأقسام الأساسية الأخرى عدا القسم الخاص بالنظام Linux , وهذا لأنه مصمم بحيث يستطيع قراءة الأقسام الخاصة بالأنظمة الأقدم منه .

القسم النشط :

عندما يحوي محرك القرص الصلب عدة أقسام أساسية يحوي كل منهما نظام تشغيل , عندها يقوم MBR بالبحث عن نظام التشغيل الموجود على القسم النشط, وكما ذكرنا سابقاً فإن قسماً أساسياً واحداً فقط يكون نشطاً .

عندما يعمل برنامج system commander تظهر شاشة تسأل عن القسم لأساسي الذي نريد جعله نشطاً . هذا جيد من أجل الأنظمة التي تحتوي على عدة أقسام أساسية , لكن ما الذي يحدث إذا كان هناك قسم أساسي وحيد ؟حسناً , يجب عند إنشاء القسم تعيينه على أنه نشط باستخدام برمجية التقسيم . هذه العملية ضرورية حتى وإن كان لديك قسم أساسي واحد فقط . يجب أن يحوي محرك القرص الصلب على قسم نشط لكي يتم الإقلاع منه .

القسم الموسع :

يمكن أن يحوي محرك القرص الصلب على النوع الآخر لأقسام القرص وهو القسم الموسع (Extended Partition) . إذا احتوى محرك القرص الصلب على قسم موسع فإنه يأخذ إحدى المناطق المخصصة للأقسام الأساسية , أي عندها يمكن إنشاء ثلاثة أقسام أساسية فقط .يعد إنشاء قسم موسع في محرك القرص الصلب عملية اختيارية تماماً . وهناك عدة أنظمة لا تستخدم الأقسام الموسعة . فهناك بعض المحركات مقسمة إلى قسم أساسي كبير واحد فقط , طبعاً لا مشكلة في ذلك . تتميز الأقسام الموسعة بطريقة تعاملها مع حروف المحركات . فعند إنشاء قسم أساسي فإنه يأخذ حرف المحرك .لكن عند إنشاء قسم موسع فإنه لا يأخذ حرف محرف بشكل آلي , بل يتم تقسيم هذا القسم إلى محركات منطقية ضمن القسم الموسع (طبعاً العدد محدود بعدد الحروف الأبجدية في أنظمة windows 9x) أي يمكن إنشاء 24 محرك منطقي في أي نظام (تذكر أن الحرفين A,B محجوزين للمحركات المرنة). يمكن تقسيم القسم الموسع إلى عدة محركات منطقية أو إلى محرك منطقي واحد , حسب الحاجة , كما يمكنك تحديد حجم كل محرك منطقي كما تريد .لكن تخلق هذه المرونة مشكلة صغيرة , وخاصة مع الأشخاص المبتدئين . وبما أن القسم الموسع المنشأ حديثاًلا يحوي بعد أي محركات منطقية , لذلك يتطلب العمل مع الأقسام الموسعة إنشاء القسم الموسع ثم إنشاء المحركات المنطقية ضمن هذا القسم . غالباً ما يخطئ التقني المبتدئ في هاتين الخطوتين , فهو ينسى إنشاء محركات منطقية , ثم يحتار لماذا لا تظهر حروف المحركات في جهاز الكمبيوتر بعد الانتهاء من التقسيم .

عملية التركيب :

من الجيد إمكانية تخصيص مساحة أكبر لوحدة التخزين عندما يبدأ المحرك الأساسي بالامتلاء وإذا لم يكن بالإمكان إضافة مساحة إلى هذا المحرك عندها يكون الخيار هو استبداله بمحرك آخر فنقطة التحميل هي محرك يعمل كمجلد محمل في مجلد آخر وللقيام بذلك أضيفي محركا فيزيائياً أخر. ثم أنشئي فيه وحدة تخزين لكن بدلا من أن تعطيه حرف محرك حملي وحدة التخزين هذه إلى مجلد في المحرك الأساسي والتي ستبدو عندها عبارة عن مجلد جديد ليس إلا . يمكن تثبيت البرامج في هذا المجلد ويمكنك استعماله لتخزين ملفات البيانات أو إجراء نسخ احتياطي لملفات النظام وبالتالي فإن محرك القرص الصلب قام بتوسيع حجم المحرك الأساسي.

الهدف منها هو ضم أقسام الهارد الغير مرئية بالنسبة للمستخدم لتكون متاحة ومرئية بالنسبة له ، بمعنى يوجد لديك نظامان تشغيل أحدهما نظام التشغيل ويندوز والآخر نظام التشغيل لينوكس والآن انت تعمل على نظام التشغيل لينوكس ولديك على القرص الخاص بك أربعة بارتشنات مختلفة اثنان منهم بنظام ملفات

EXT3 وواحد بنظام ملفاتFAT32

NTFSو FAT32 وعندما قمت بالدخول إلى نظام التشغيل LINUX حدث شيء غريب ألا وهو أن الأقسام الخاصة بنظاميّ الملفات NTFS والآخر

لا يمكنك قراءة أيٍ منهما فما العمل ؟

اختفت الأقسام بلا رجعة ؟ هل حدث خطأ ما ؟؟ الإجابة لا ...لا تقلقي فكل ما في الأمر أن نظام التشغيل بشكل تلقائى لا يرى إلا القسم الذى تم تثبيته فيه ولكي تتمكن من العمل على باقي الأقسام لابد من عملية التركيب التي ذكرناها سابقا ولكن كيف لنا أن نقوم بهذه العملية ؟؟

أولاً وبشكل بسيط جدا لابد من التعرف على تلك الأقسام وأين توجد.بمعنى ، يتعامل نظام التشغيل لينوكس أقسام القرص الصلب بشكل مختلف تماما عن نظام التشغيل ويندوز ,بمعنى على ويندوز نظام الهارد تحمل الحروف التاليه (C,D,E,F,G,H,I) وهكذا ولكن نظام التشغيل لينوكس يتعامل مع الأقسام بشكل مختلف فمسمياته بالنسبة للقرص من نوع ATA تكون هكذا hda1 , hda5 , hda6 , hda7 , hda8.

نتقل الآن إلى كيفية عرض تلك الأقسام partition , يكون من خلال الامر التالى فى وضعية الرووت

```
$fdisk -l

  Device  Boot          Start                End
Blocks    Id  System
/dev/hda1                    1                388
2933248+  83  Linux
/dev/hda2                  389                557
1277640   5   Extended
/dev/hda5                  389                557
1277608+  b   W95 FAT32
```

ولتمييز الأقسام اللي عمل لها عملية التركيب من التي لم نعمل لها ، نفتح الطرفية لدينا (shell) ونكتب فيه :

```
$mount
```

```
/dev/hda1 on / type ext3 (rw,errors=remount-ro)
tmpfs on /lib/init/rw type tmpfs (rw,nosuid,mode=0755)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
procbususb on /proc/bus/usb type usbfs (rw)
udev on /dev type tmpfs (rw,mode=0755)
tmpfs on /dev/shm type tmpfs (rw,nosuid,nodev)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=620)
nfsd on /proc/fs/nfsd type nfsd (rw)
rpc_pipefs on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
```

من السابق اتضح لنا أ ن hda5 لم يخضع لعملية التركيب , ونريد الان عمل له ماونت ,اذن نتبع الاتي :

نقطة التركيب:

بمعنى لابد من إيجاد رابط لهذا الجزء المراد عمل التركيب له mount point مع المكان الاصلي لهذا الجزء.

وفى المثال السابق المكان الأصلى للبارتشن فى المسار/dev/hda5

نقوم بإنشاء مجلد آخر تحت أى مسار على النظام لربط الجزء بنقطة الضم أو ال mount point

يوجد مساران على اى توزيعة لينوكس من المفضل إنشاء نقطة الـــ mount لها الا وهما : /mnt ,/media

الآن نقوم بإنشاء نقطه للضم على أي من المسارين الذين ذكرتهما سابقا ولا بد من القيام بهذه العملية على وضعية الـــ root .

```
$mkdir /mnt/xxxxx
```

ملفات الان نقوم بتنفيذ أمر التركيب للجزء hda5 ويكون ذلك من خلال الأمر التالى :

```
$mount -t vfat /dev/hda5 /mnt/xxxxx
```

11.2.3 نظام الملفات التخيلي:

تعريفه :

هو طبقة في لب النظام (kernel) التابع لنظام ملفات معين و تقوم بتحولها إلى أخرى يفهمها نظام ملفات آخر وتتعامل مع الـــ system calls

و بالتالي توفر طريقة تعامل متعارف عليها بين أنظمة الملفات المختلفة

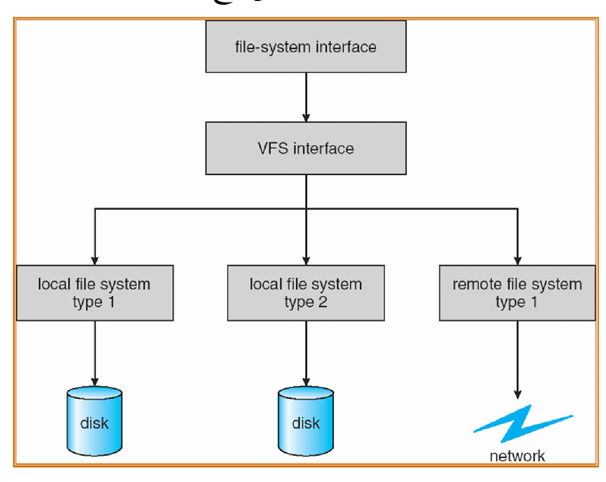
يتيح نظام الملفات التخيلي الوصول إلى وحدات تخزين محلية أو على شبكة, دون أن ينتبه المستخدم إلى وجود فروقات في أنظمة الملفات ، وبالتالي يستطيع أن يمد جسر بين أنظمة ملفات windows, mac os & linux

بحيث يوفر إمكانية فتح ملف تابع لنظام ملفات معين دون الحاجة لمعرفة نوعه.

فأي أمر صادر من الطبقة العليا (النظام الحالي) يقوم بتحويله إلى أمر يفهمه النظام في الطبقة السفلى, فهو كواجهة بين أنظمة الملفات.

أحد أنظمة الملفات التخيلية والمستخدمة مع النسخ الأولى من الـــ unix ، هو نظام sun os من شركة sun microsystems

عبر الشبكة بشفافية تامة NFSالمحلية و أنظمة UFS التعامل مع أنظمةunix و قد أتاحت لـــ



المصادر :

http://web.mit.edu/tytso/www/linux/ext2intro.html
http://209.85.129.104/search?q=cache:VkZOKtxXbiwJ:wapedia.mobi/en/Virtual_file_system+virtual+file+systems&hl=ar&ct=clnk&cd=31&gl=sa
Operating Systems Concepts

كتبه :

منيرة السريّع

زاهيه الحربي

11.3 ***Directory implementation***

القائمة الخطية (***Linear list***)

اسهل طريقة لتنفيذ الدليل هي استخدام القائمة الخطية لاسماء الملفات مع مؤشرات لبيانات الــ block في الذاكرة وهي سهلة للبرنامج ولكن تستغرق وقتاً طويلاً لتنفيذه

من مميزاتها انها سهلة البرمجة ولكن فيها إضاعة واستهلاك للوقت , فعند تكوين ملف جديد لابد من عمل بحث على الدليل للتأكد من عدم وجود ملف بنفس الاسم بعدها يُضاف هذا الملف الجديد لنهاية الدليل. وعند حذف ملف لابد من عمل بحث عن الملف المطلوب ثم حذفه ، وفي أسوء الأحوال عندما يكون الملف في آخر الدليل فإننا نحتاج إلى المرور على جميع الملفات ..

ولاعادة استخدام مدخُلات الدليل نستطيع استخدام عدة طرق :

0 وضع علامة (mark) عند المُدخل كمعطل او غير مستخدم اما باسناده الى اسم معين كجميع الاسماء الفارغة او ببت المستخدم\غيرمستخدم لكل مُدخل

0 او نضعه في قائمة مُدخلات الدليل الحرة

0 او نسخ آخر مُدخل في الدليل الى مساحة حرة وهذا يقلل من طول الدليل

والقائمة الخطية تستخدم لتقليل الوقت المستخدم لحذف ملف.

ولكن عيب القائمة الخطية هو البحث عن ملف معين في الدليل .

ومعلومات الدليل تستخدم بكثرة وقد يلاحظ المستخدم بطئها عند المعالجة

<u>***جداول المزج (Hash Table)***</u>

قائمة خطيه بhash data structure

من مميزاتها أنها تقلل من وقت البحث حيث تأخذ الجداول قيمه محسوبة من اسم الملف عن طريقها ترجع مؤشر يأشر على مكان الملف في القائمة, كما أن عمليات الإضافة والحذف أسهل وأسرع.

ومن عيوبها :

أن الجداول لها حجم محدود ، وموقع الملفات يعتمد على حجم الجدول ،

فإذا امتلئ الجدول فإنه يجب انشاء جدول جديد بحجم أكبر كما يجب إعادة حساب موقع جميع الملفات ، وهذا فيه استهلاك كبير للوقت .

كما أنه قد يحدث تصادم بان يُعطى ملفان مختلفان نفس المكان.

لحل التصادم راح يكون كل مدخل في الجدول مرتبط بقائمة متصلة (linked list) وأي مدخل جديد سوف يضاف للقائمة لموجودة في المكان المناسب لاسم الملف.

كتبه :
اروى العريفي
أماني الشهراني

11.4 طرق الحجز :

واحده من أصعب المشاكل التي تواجهنا في الملفات هي كيفية حجز مساحه للملفات ..

من ناحيتين:-

الأولى أن مساحه القرص الصلب يجب أن تكون مستغله تمام الاستغلال وبكفاءة عالية.

والناحية الأخرى هي سرعه البحث والدخول للملف المطلوب بسرعة.

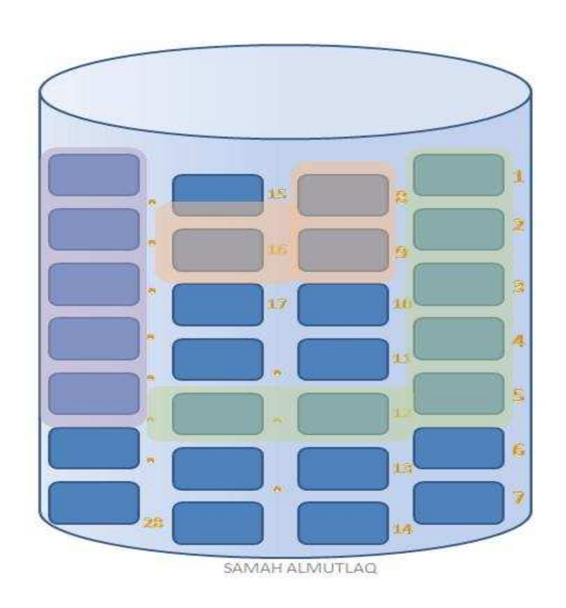
وبعد هذا المقدمة المقتضبة سنتطرق الآن إلى الثلاث أنواع الرئيسية لطرق الحجز:-

الطريقة الأولى هي الحجز المتواصل الغير منقطع:- (Contiguous Allocation)

أي أن يستولي كل ملف على مجموعه متتالية من العناوين (address) أي على مجموعه وحدات (block)

كما هو موضح بالرسم حيث إن كل لون هو عبارة عن أماكن محجوزة لملف

والتي باللون الأزرق هي أماكن فارغة غير محجوزة:-



ونلاحظ أن العناوين بالقرص(disk address) مرتبة ترتيب خطي بالقرص (disk) ,وتتميز هذه الطريقة للحجز في أن الإزاحات المتطلبة للعبور المتتابع (accessing contiguous) لحجز الملفات تقل وهذا متطلب من متطلبات الحجز التي ذكرناهما مسبقا وهما السرعة والمساحة.

أما الوصول إلى ملف معين فيتم ذلك بسهوله فنستخدم فقط عنوان البداية للوحدة الأولى(block address of first) لهذا الملف والطول له (أي أن هذا الملف على كم وحده يسيطر؟)

كون الملف يحجز وحدات(blocks) متتابعة فهذا لا يمنع الدخول أو العبور العشوائي لهذه الوحدات (blocks) وهذه من حسنات طريقه الحجز المتتابع.

لكن الصعوبة في هذا الطريقة تظهر في إيجاد مساحه جديدة للملفات الجديدة فلو فرضنا إن الملف الذي سوف يُنشأ سيوضع في س من الوحدات (blocks) فان نظام التشغيل يجب إن يبحث عن س من الأماكن الفارغة المتتابعة.

التجزئة الأولى (First-fit), والتجزئة الأسوأ (worst-fit) ,والتجزئة الأفضل (best -fit) هي الاستراتيجيات الأكثر شيوعا المستخدمة لاختيار المكان الفارغ المناسب من بين مجموعه من الأماكن الفارغة لكن جميعها لا تناسب من ناحية الاستخدام والاستغلال الأمثل للقرص الصلب ولكن الأنسب من ناحية السرعة هنا هي التجزئة الأولى(First-fit) وهي استراتيجيات سبق التطرق إليها.

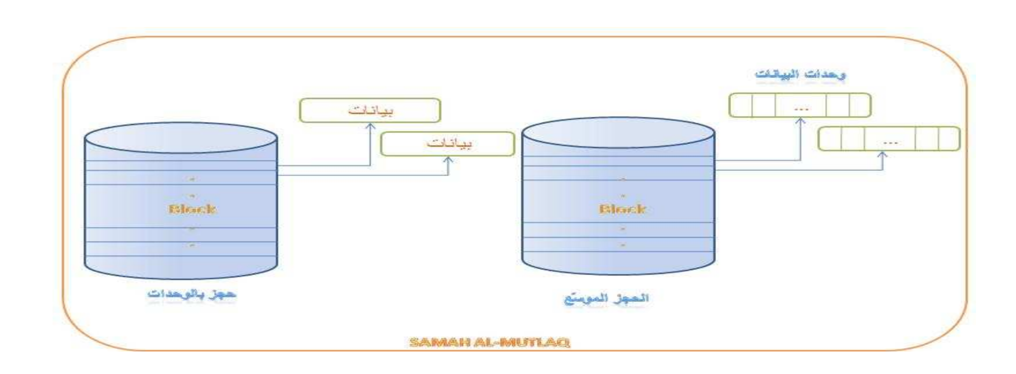
عيوبها:

* استخدام هذه الطريقة يتسبب في تضييع مساحات على القرص الصلب وذلك لأنها تعاني من external fragmentation, أي أنه مع حجز أماكن للملفات وحذفها سينتج لدينا مساحات فارغة متفرقة على القرص الصلب, بحيث تكون هذه المساحات متفرقة و الواحدة منها تكون غير كافيه لتخزين أجزاء ملف متتابعة وبالتالي تستمر هذه المساحات فارغة بدون استخدام.

*تحد هذه الطريقة من زيادة أحجام الملفات, أي أن الملف عندما يحجز مساحة فإن هذه المساحة تكون غير قابلة للزيادة وذلك لأن الزيادة يجب أن تكون تابعة لأجزاء الملف , أي بعدها مباشرة, وقد يكون المكان الذي بعدها محجوزاً لملف آخر وبالتالي لن يستطيع الملف زيادة حجمه!

هناك إصدار جديد معدّل على الطريقة الأولى (الحجز المتواصل الغير منقطع) وهو حل لمحدودية المكان وهو عبارة عن وحدات (blocks) تحتوي على مجموعات كبيره في وقت معين والتي ترغم أن يكون الحجز متتابع.

كما في الرسم فيما يلي:-

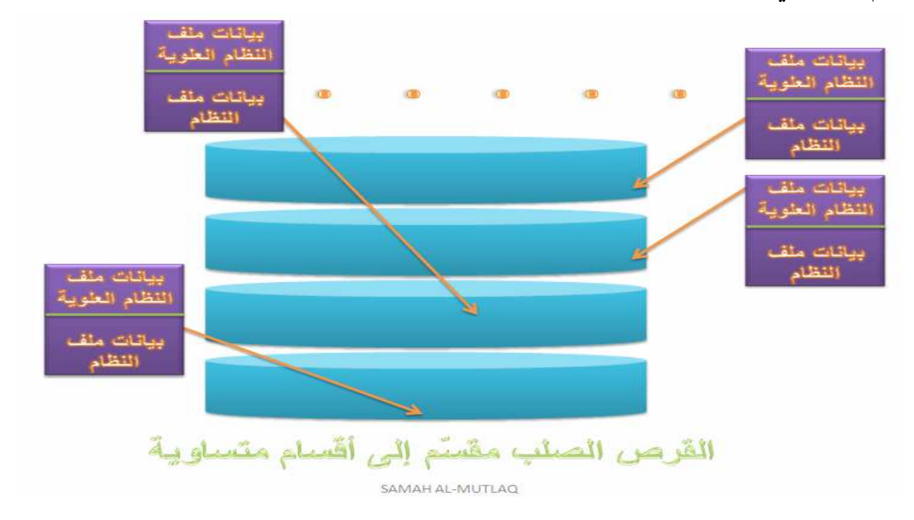


آلية عملها انه عندما يُكتب ملف ما تقوم بحجز عدد كبير من الوحدات (blocks) وتنقسم البيانات بالوا حدت إلى قسمين:-

(metadata) هي البيانات العلوية و ُتكتب أول ما يُنشأ الملف.

(Subsequent) وهي البيانات السفلية وتّكتب مع أول حجز متوسع للوحدات.

كما في الرسم فيما يلي:-



ملحوظة:-

لا نحتاج بيانات علوية إضافية(metadata) إلا بعد الحجز المتوسع التالي.

الطريقة الثانية من طرق الحجز هي الحجز المترابط:- (Linked Allocation)

في الحجز المرتبط كل ملف هو قائمه مرتبطة (linked list) من وحدات التخزين (Disk blocks) لكنها ليست متتابعة فعلياً في وحده التخزين وإنما بأماكن متفرقة.

وفيه يوجد دليل يحتوي على مؤشر لأول (واختياري آخر) وحدة من الملف وقيمة المؤشر الابتدائية تكون عديمة (null) وهي تكون لآخر طرف (node)في القائمة.

وتعتمد الكتابة على الملف في هذه الطريقة من الحجز على الكتابة في أول وحدة (block) فارغة بحيث تُربط هذه الوحدة الجديدة بالقائمة (linked list)فيما بعد.

ولقراءة الملف فإن المؤشر يتنقل بين الوحدات(blocks) من وحدة لأخرى .

ومما يميز هذه الطريقة هو انه لا يوجد مشكلة التجزئة الخارجية(external fragmentation) ,فأي وحدة(block) فارغة يمكن أن تستخدم ,وكما انه يتميز عن الطريقة الأولى في انه لا يحتاج معرفه حجم معين للملف قبل إنشائه فالملف يمكن أن ينمو على قدر ما يوجد وحدات فارغة (blocks).

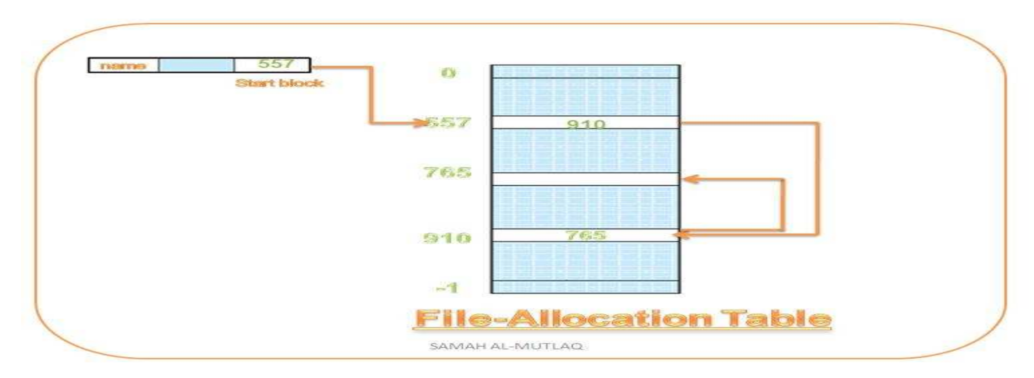
لكن سلبية الحجز المرتبط تكمن في انه غير فعّال للعبور(access) المباشر أو للعبور العشوائي فهو فعّال فقط بالعبور(access) التسلسلي ,فلو أردنا وحدة معينه على سبيل المثال فانه سيضطر إلى المرور على كل الملفات من البداية للنهاية وهذا مكلف للوقت فهو يتطلب قراءه وحدة التخزين .

هناك مشكلة أخرى في هذه الطريقة تكمن في المساحة المستهلكة لتخزين المؤشرات, أي أن المؤشرات سوف تستهلك مساحة من القرص كان من الممكن أن نستغلها لتخزين بيانات فيها.

ولا نغفل أن نذكر سلبية أخرى إلا وهي انه لا يمكن الوثوق به و الاعتماد عليه(reliability) لان الملفات مرتبطة ، بعضها البعض عن طريق مؤشرات فلو حدث خطأ ما وفقدنا مؤشر(pointer) فهذا الخطأ يؤدي بنا إلى أن يكون مؤشر الملف المتضرر يؤشر على مكان فارغ أو على ملف آخر.

الجدير بالذكر أن هناك تغير طرأ على الحجز المرتبط باستخدام جدول حجز الملفات(File-Allocation Table) حيث انه في بداية كل حيّز (volume) يوضع بالجدول بدايات كل الوحدات(blocks) عن طريق رقمها(أي الوحدة) حيث ان هذه البدايات مرتبطة في الجدول بعضها البعض كالسلسلة.

كما هو موضح بهذا الرسم:-



وأخيرا الطريقة الثالثة وهي الحجز المفهرس:- (Indexed Allocation)

طريقة الحجز المفهرس حل للمشكلتين السابقتين بحيث توضع جميع المؤشرات مع بعضها البعض في وحدة واحدة (block) وتسمى الفهرس (index)

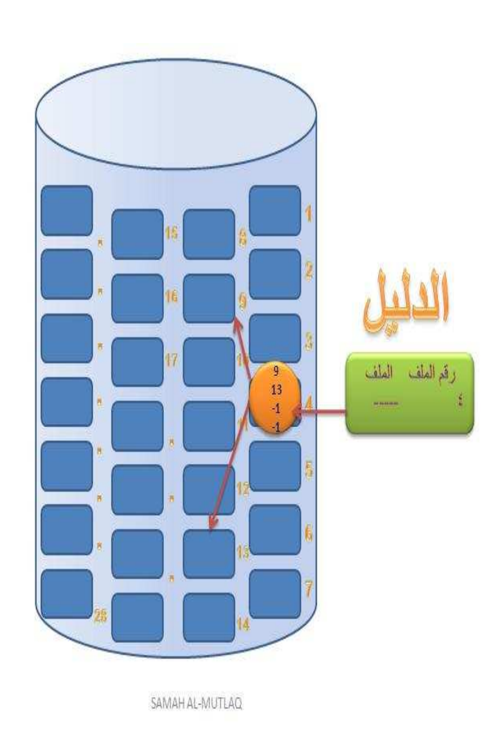
بحيث أن كل ملف له فهرس لوحدة التخزين الخاصة فيه والتي تؤشر على قطاع معين في قرص التخزين

ولقراءة القطاع رقم س من الملف, فالمؤشر عند رقم س في الفهرس ليجد القطاع المطلوب

ومما يميز هذا النوع هو انه يدعم الدخول المباشر(direct access) للوحدة المطلوبة ,بدون عناء التجزؤ الخارجي(external fragmentation) فأي مكان فارغ على القرص الصلب يمكن أن يوضع فيه ملفات يدلنا على مكانها الفهرس

ولكنها طريقة غير فعّالة إذا كان هناك ملفات قليلة ، حيث تعاني هذه الطريقة من مشكلة المساحة المستخدمة في تخزين المؤشرات في index block , أي أننا قد نحجز مساحة لهذا الجزء ونضع بداخله مؤشر واحد أول اثنين فقط! وبالتالي نكون قد حجزنا كل هذه المساحة بدون استغلالها بشكل جيد.

كما هو موضح بهذا الرسم:-



المصادر :

http://filesystems.palconit.com/filesystems-file-allocation-methods.html

http://www.people.fas.harvard.edu/~lib215/lectures/lect_p/lect04/6_Extras/bar_fs/index3.htm

Silberschatz, Galvin, Gagne: Operating System Concepts

كتبته

سماح المطلق

11.5 إدارة المساحة الحرة "***management Free space***":

بما أن سعة القرص محدودة، من الضروريُ إعادة استخدام المساحة الفارغة الناتجة مِنْ الملفاتِ المَحْذُوفةِ للملفاتِ الجديدةِ.

لِمُتَابَعَة مساحة القرص الحرة، يحتفظ النظام لديه قائمة بالمساحات الفارغة في القرص تسمى بـــ قائمة المساحات الحرة (Free space list).

تُسجّلُ قائمةُ المساحات الحرة كُلّ كُتَل القرص الحرة (free disk blocks)

(بمعنى آخر. ،الكتل التي لم تخصص للملفات)

لإنْشاء ملف، يجب البحث في قائمة المساحات الحرة عن المساحة المناسبة المطلوبة لتخزين الملف، ويتم تُخصّيصُ تلك المساحة لهذا ملف جديد. ثم يتم إزالة هذه المساحة من قائمة المساحات الحرة. وبالمقابل عند حذف ملف، فإن المساحة التي كانت محجوزة له تُضافُ إلى قائمةِ المساحات الحرة.

تستخدم أنظمة التشغيل المعروفة عدة طرق لتنفيذ قائمة المساحات الحرة منها:

1. متجه البت "bit vector"
2. القائمة المرتبطة "linked list":
3. التجمع "Grouping"
4. التعداد " counting "

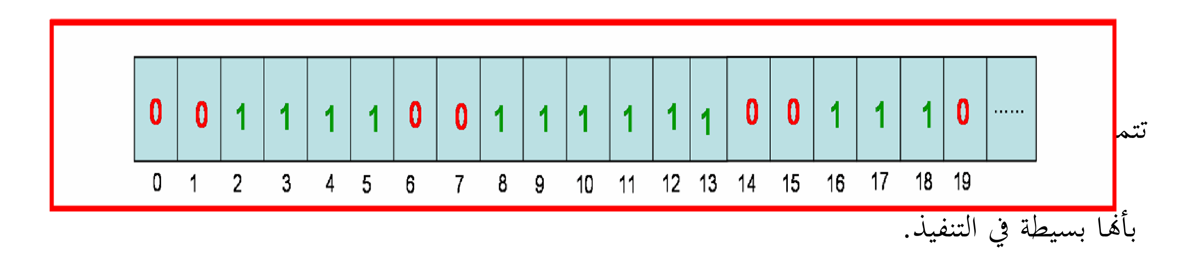
طريقة متجه البت: "bit vector"

في هذه الطريقة نجد أن كل كتلة "Block" يتم تمثيلها بـــ بت واحد " one bit".

إذا كانت الكتلة فارغة فإن قيمة هذه البت تساوي (0) ، وإذا كانت الكتلة مُخَصَّصةُ فإن قيمة هذه البت تساوي (1) (بعض أنظمة التشغيل تعكس القيم السابقة ، بمعنى أنها تعتبر الـــ 1 محجوز و الــ0 فارغ)

مثال:

إذا كان لدينا قرص و يحتوي على blocks فارغة (2,3,4,5,8,9,10,11,12,13,1617,18) و باقي ال blocks محجوزة يكون تمثيل الـــ bit map



تتم

بأنها بسيطة في التنفيذ.

و فعالة لإيجاد أول كتلة حرة، أو عدد من الكتل الحرة المتتالية حيث أن معظم المعالجات تدعم ما يسمى ب"bit-manipulation instruction"

المساوئ:

غير مناسبة إلا إذا كان كامل مخطط البتات موجود في الذاكرةِ الرئيسية بالنسبة للكتل الأكثر استخداما.

إبْقائه في الذاكرة الرئيسية ممكن في الأقراص الصغيرةِ مثل الحاسبات الصغرى microcomputers، لكن غير ممكن في الحاسبات الكبيرة.

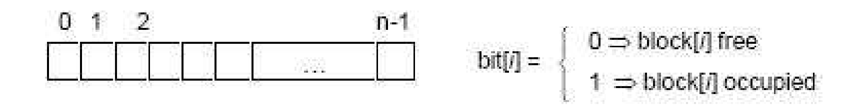
مثال:

نفرض أن حجم ال block يساوي 2Kbyte و حجم القرص يساوي 1 Gbyte و بالتالي يكون حجم ال bit vector يساوي 1GB/2KB=32KB

طريقة حساب عدد الكتل "blocks" :

يتم حساب مكان أول block فارغ بالبحث (تسلسلي)عن أول word في ال bit map تحتوي block فارغة (bit 1)

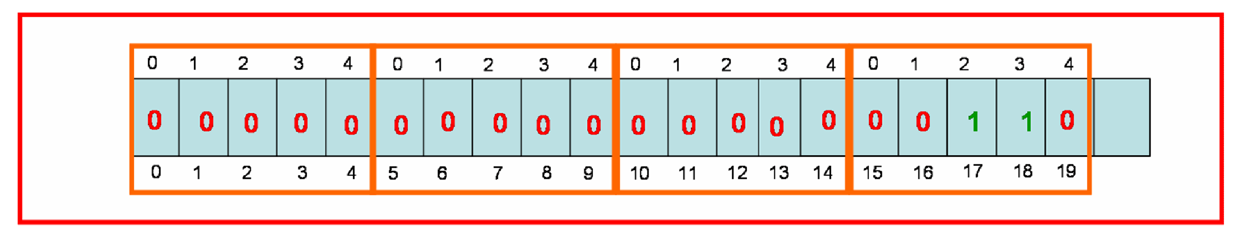
رقم أول ال block فارغ = (عدد ال bits في كل word * عدد ال words التي تحتوي أصفار)+ offset لأول block فارغ



مثال:

نفرض أن ال word تساوي 5 bits

وأن الـــ bit map كما في الشكل التالي :



فإن رقم أول block فارغ =(5 * 3) + 2 = 17

طريقةالقائمة المرتبطة"linked list":

يتم ربط جميع المساحات الحرة معا على القرص ، من خلال إبقاء المؤشر على أول كتلة في موقع خاص على القرص واستخدامه مخبأ في الذاكرة .
الكتلة الأولى تحتوي على مؤشر يؤدي الى الكتلة التالية ، والتي بدورها تحتوي على مؤشر للكتلة القادمة ، وهكذا..

مثال:

في المثالِ السابقِ, المؤشر يُمْكِنُ أَن يشير عل الكتلة الحرة رقم 2، ككتلة حرة أولى. وبالتالي هذه الكتلة تحتوي على مؤشر يشير على الكتلة الحرة رقم 3 التي تحتوي هي بدورها على مؤشر يشير على الكتله الحرة رقم 4 والتي تشيرُ لكتلة 5 التي تشيرُ لكتلة رقم 8، وهكذا.

ميزة هذه الطريقة أنها:

* لا يتم فيها تضييع للمساحات (مجرد مؤشر لأول كتلة)

و من عيوبها انها:

* ليست فعالة في اجتياز القائمة.

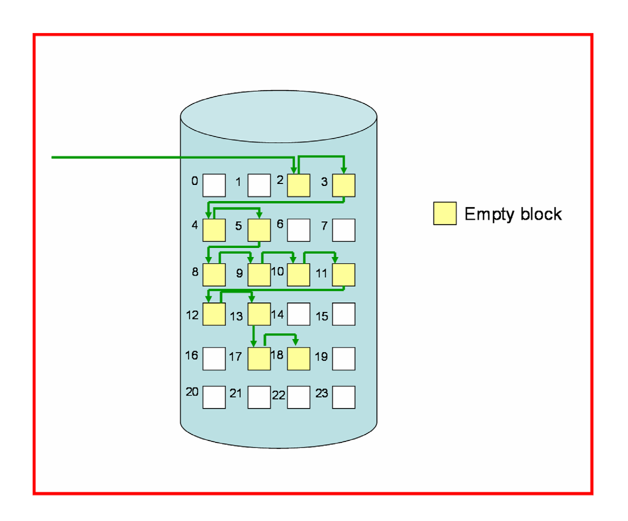
لأنه كي نصل لكتله معينه، يستلزم قراءه كل الكتل التي قبلها، وهذا يتطلب وقت إدخال/إخراج كبير وهدر للوقت.

* لا يمكن الحصول بسهولة على مساحات متتالية.

* تستخدم بكثافه الكتلة الامامية " block front" اكبر من الكتلة الخلفية.

مثال:

إذا كان لدينا قرص و يحتوي على blocks فارغة (2,3,4,5,8,9,10,11,12,13,17,18) و باقي ال blocks محجوزه يكون تمثيل ال linked list



طريقة التجمع "Grouping":

* الكتلة الحرة تتضمن عدد من المؤشرات الموصلة إلى الكتل الحرة.

* الكتلة الأخيرة تتضمن بدورها عدد من المؤشرات المؤدية إلى كتل حرة أخرى.

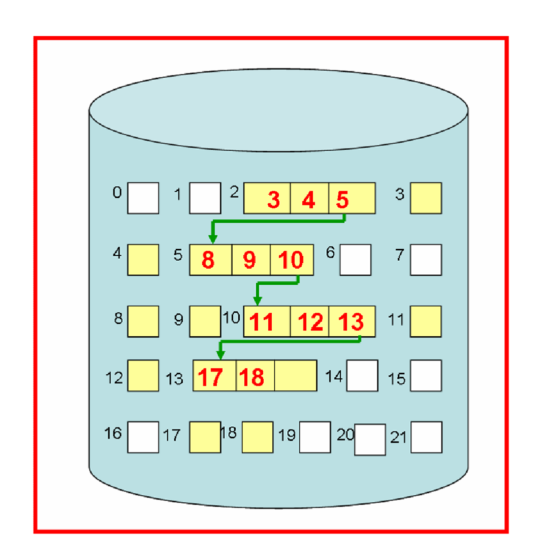
تقوم هذه الطريقة بتخزين عناوين ال blocks الفارغة n في أول block فارغ ،نلاحظ في كل مجموعة أن ال blocks الفارغة فعليا n-1 (آخر عنوان هو عنوان المجوعة التي تليها)

مميزات هذه الطريقة :

يمكن الحصول عن مجموعة من عناوين ال blocks الفارغة بسرعة و سهولة

مثال:

إذا كان لدينا قرص و يحتوي على blocks فارغة (2,3,4,5,8,9,10,11,12,13,17,18) و باقي ال blocks محجوزه يكون تمثيل ال grouping (n=3)



طريقة التعداد "counting":

هذه الطريقة تَستفيد من حقيقة أن عِدّة كُتَل متجاورة (contiguous blocks)قَدْ تُخصّصُ أَو تُحرّرُ بشكل آني، خصوصاً عند استخدام طريقة التخصيص المتجاور (contiguous allocation)،لذلك بدلاً مِنْ إبْقاء قائمة بالمساحات الحرة في القرص، فقط يتم إبقاء عنوان الكتلةِ الفارغة الأولى وعدد n مِنْ الكُتَلِ الفارغة المجاورة لها(التي تْلي الكتلةَ الأولى).

بالتالي في قائمة المساحات الحرة يتم إدخال عنوان في القرص وعدد معين فقط.

مميزات هذه الطريقة:

بالرغم من أن كُلّ مدخل في القائمة (عنوان+عدد) يَتطلّبُ مساحة أكثرَ مِنْ عنوان بسيط في قرصِ، هذه القائمة سَتَكُونُ أقصر عموما إذا كان العدد المدخل اكبر من 1.

مثال

إذا كان لدينا قرص و يحتوي على blocks فارغة (2,3,4,5,8,9,10,11,12,13,17,18) و باقي ال blocks محجوزه يكون تمثيل ال counting

free block list

Address	Count
2	11
3	10
4	9
5	8
8	7
9	6
10	5
11	4
12	3
13	2

المصادر :

www.ece.iupui.edu/~dskim/Classes/ECE408/ln-os-ch11%20File-System

http://filesystems.palconit.com/filesystems-free-space-management.html

http://blog.pixnet.net/nixchun/post/7989490

Operating System Concepts

بواسطة :

- Khadija akherfi.
- لمياء المقبيل .
- خلود صالح الرومي .

11.6 الآداء والفعالية :

Buffer Cache / page Cache / Unified Buffer Cache

الأنظمة الحديثة تتيح الوصول إلى بيانات ملف في القرص (نظام الملفات) بطريقتين:

1. Memory mapping مثل mmap()

والتي تعالج بواسطة نظام الذاكرة الافتراضية (Virtual Memory subsystem)

2. I/O system calls مثل read() وَ Write()

والتي تعالج بواسطة نظام وحدات الإدخال والإخراج.(I/O subsystem)

Disk Cache

آلية لتقليل الوقت المستغرق للقراءة أو الكتابة من وإلى القرص الصلب.

أنواع Disk Cache

1. Buffer Cache

Buffer cache وسيط بين I/O system calls (أوامر القراءة والكتابة) و القرص (نظام الملفات).

في السابق كان يخصص جزء من الذاكرة؛ لـِ Buffer Cache، وحالياً يوجد كجزء من القرص الصلب.

وظيفته:

الاحتفاظ بالبيانات المقروءة حالياً و البيانات المجاورة لها في القرص، في حالة لو طُلبت مستقبلاً. كما يحتفظ بالبيانات المكتوبة حالياً لفترة ثم تنقل إلى القرص.

آلية القراءة:

عند القراءة من القرص، تتحرك ذراع القرص حتى يصل رأس القراءة إلى المسار المطلوب، وبعد فترة زمنية يقوم الرأس بجمع bits. وعادة لا تكون Sectors المقروءة في البداية هي الوحيدة المطلوبة من قبل نظام التشغيل فقد يحتاج إلى زيادة لاحقاً ؛ وبسبب ذلك يتم الاحتفاظ بالــSectors (Blocks)المقروءة والمجاورة لها في Buffer Cache فتكون جاهزة في حالة طلبها نظام التشغيل لاحقاً.

لماذاBuffer Cache:

لأن سرعة وحدات الإدخال والإخراج أسرع من نقل Bits من وإلى القرص، تستخدم Buffer Cache ليتسنى للاثنين العمل بأقصى سرعة ممكنة دون حدوث تأخير أو إرباك.

2. Page Cache

تنقل بيانات الملف المستخدم والمجاورة لها باستخدام تقنية الذاكرة الافتراضية و تحتفظ بها كــ Pages بدلاً من Block، أي أنها تتعامل مع العنوان الافتراضي وهذا أسرع من استخدام العنوان الفعلي لــ blocks .

3. Two cache model

تستخدم Buffer Cache وPage Cache معاً

- في حالة استخدام Buffer Cache (أي عند إصدار I/O system calls)

عند القراءة من ملف تنقل بياناته من القرص إلى الـــ Buffer Cache ومنها إلى التطبيق الذي طلب تلك البيانات.

وفي الكتابة يحدث العكس، حيث تعدل البيانات في Buffer Cache ثم تنقل إلى القرص فيما بعد.

- في حالة استخدام page cache(أي memory mapping I/O)

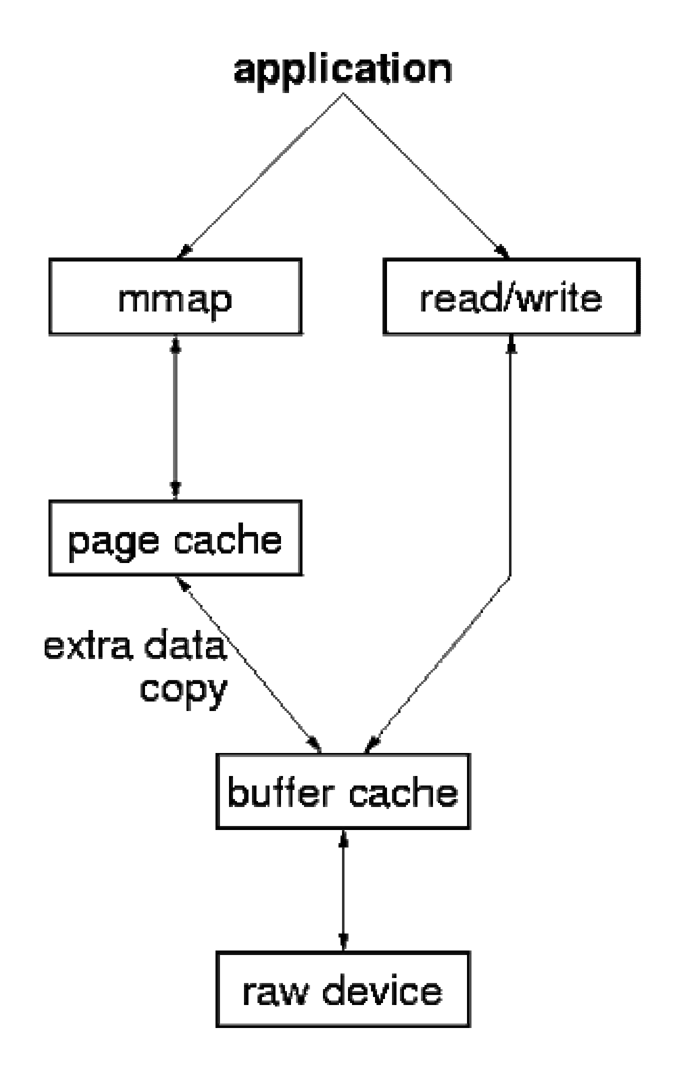
تنقل البيانات من القرص إلى Buffer Cache ولأن الأخير لا يتعامل مع نظام الذاكرة الافتراضية؛ يتوجب في كل مرة تنقل فيها البيانات أن تنسخ أيضاً في page cache لأنه يعالج بواسطة نظام الذاكرة الافتراضية وبهذا يمكن نقل البيانات إلى التطبيق الذي طلبها.

وعكس ذلك في الكتابة، حيث يتم تعدل البيانات في page cache ثم تنسخ إلى Buffer cache لتكتب فيما بعد على القرص.

إن أداء هذه العمليات بطيء كما أن عملية النقل للبيانات مرتين مكلفة من حيث:

1. استهلاك مساحة الذاكرة (ضعفي حجم بيانات الملف)؛ وبذلك تقل مساحة الذاكرة المتاحة للتطبيقات.
2. إضاعة وقت المعالج (فترة نقل البيانات بين الـــ page cache و Buffer cache)

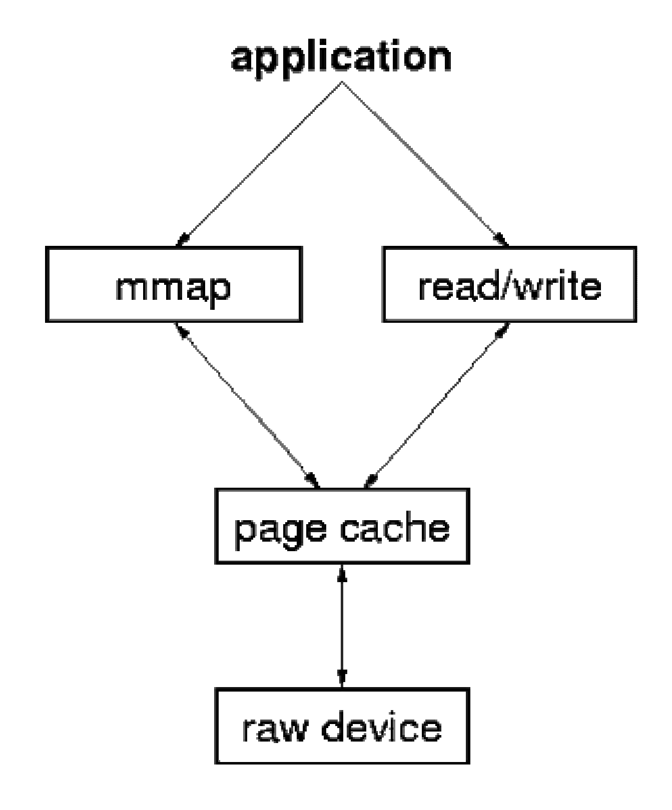
كما أن وجود نسختين من نفس البيانات قد يؤدي إلى حدوث عدم توافق (مثلاً: تطبيق يعدل على بيانات ملف في page cache وخلال ذلك يصدر تطبيق آخر أمر قراءة لنفس الملف ، سوف يقرأ من Buffer cache معلومات غير محدثة !)



ولحل هذه المشاكل تم التوصل إلى ابتكار نظام Unified Buffer Cache

4 . Unified Buffer Cache(UBC)

في هذا النظام يتم نقل بيانات الملف المطلوبة دائماً بغض النظر عن كيفية طلبها إلى page cache مباشرة بدون الحاجة إلى وجود buffer cacheوبهذا يمكن لنظام الذاكرة الافتراضية إدارات بيانات الملف مباشرة .



المصادر:

- http://searchstorage.techtarget.com/sDefinition/0,,sid5_gci211963,00.html
- http://www.faqs.org/docs/linux_admin/buffer-cache.html
- http://www.usenix.org/publications/library/proceedings/usenix2000/freenix/full_papers/silvers/silvers_html/
- Operating system concepts-6th ed(page 434-435)

الفصل التاسع:
تراكيب وحدات التخزين

تراكيب وحدات التخزين

Mass-Storage Structure

تمهيد:

في هذا الفصل , سوف نتحدث عن المستوى الأدنى من نظام الملفات وهو : تراكيب التخزين الثانوية.

في البداية سوف نصف التركيب الفيزيائي للأقراص والأشرطة المغناطيسية . ثم سنتحدث عن تهيئة القرص بشيء من الإيجاز .

بعد ذلك سوف نتحدث عن خوارزميات جدولة القرص التي تنظم ترتيب عمليات الإدخال والإخراج للقرص وذلك لتحسين أداءه . بعد ذلك سوف نتحدث عن إدارة فضاء التبديل .

أهداف الفصل :

- وصف التركيب الفيزيائي لأجهزة التخزين الثانوية و التأثير الناتج عن استخدام هذه الأجهزة.
- توضيح خصائص أداء أجهزة التخزين.
- مناقشة خدمات نظام التشغيل المُقدمَة لأجهزة التخزين , متضمنة RAID .

تم الجمع و التنسيق بحمد المولى تبارك وتعالى، إن أصبت فمن الله وحده وإن أخطأت فمن نفسي والشيطان والصلاة والسلام على أشرف الأنبياء والمرسلين. ليلى بنت علي البيشي

(l.bishy@gmail.com)

1-12 نظرة عامة عن تركيب أجهزة التخزين:

في هذا القسم نعرض نظرة عامة عن التركيب الفيزيائي لأجهزة التخزين الثانوية : القرص المغناطيسي ، والشريط المغناطيسي .

1-1-12 القرص المغناطيسي :

تتركب الأقراص المغناطيسية من أقراص مفردة أو أطباق (platters)

- وهذه الأطباق مصنوعة من الألمنيوم المغلف بمادة مغناطيسية من كلا الجانبين

و يتطلب كل طبق : رأسي قراءة / كتابة , يُتحكم بها بواسطة محرك سيرفو .

كل طبق مقسم إلى مسارات (tracks) وكل مسار مقسم إلى قطاعات **(sectors)**, .

حيث يعتبر القطاع أصغر وحدة تخزين , وهو يخزن 512 بايت .

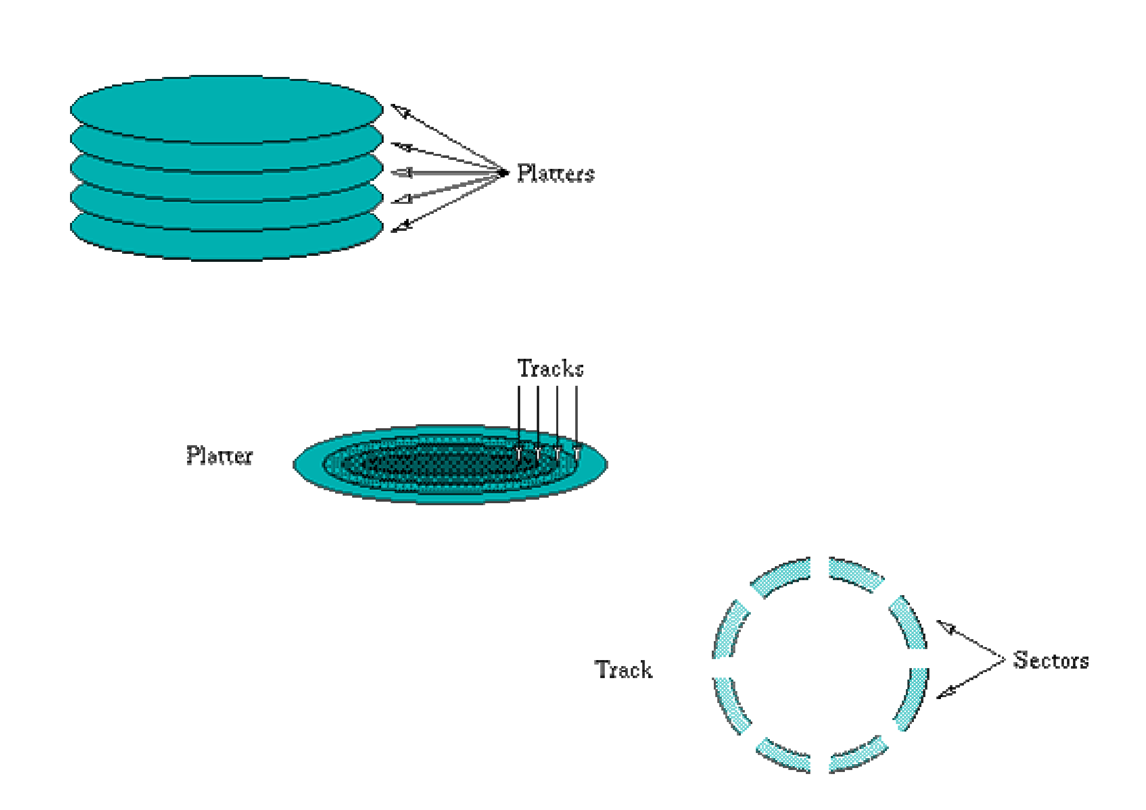
و لكي نقوم بعملية إدخال أو إخراج فإن القرص يقوم بتحريك الرأس أولاً إلى المسار ثم إلى القطاع الصحيح.

و في أكثر الأنظمة ، تكون الأذرع (arms) مرتبطة مع بعضها ؛ لكي تتحرك الرؤوس (heads)مع بعضها البعض ،

وبذلك كل رأس يمر على نفس المسار في كل سطح .

ثم تتحرك أداة القراءة والكتابة أو (رأسي القراءة / الكتابة) على سطح الأقراص حتى تحدد المسار والقطاع المطلوب.

. (cylinder)يُطلق على جميع المسارات التي لها نفس القُطر : إسطوانة



تركيب القرص المغناطيسي

يحتوي القرص المغناطيسي   على أجزاء ميكانيكية و أخرى إلكترونية . وبالمُجمل الأجزاء الميكانيكية

. **(Platters)** والأطباق   **(cylinder)** وهي التي ذكرت في الصفحة السابقة مثل الإسطوانة
)

و رأس للقراءة و الكتابة على كل سطح من أسطح الأطباق (الأقراص) .

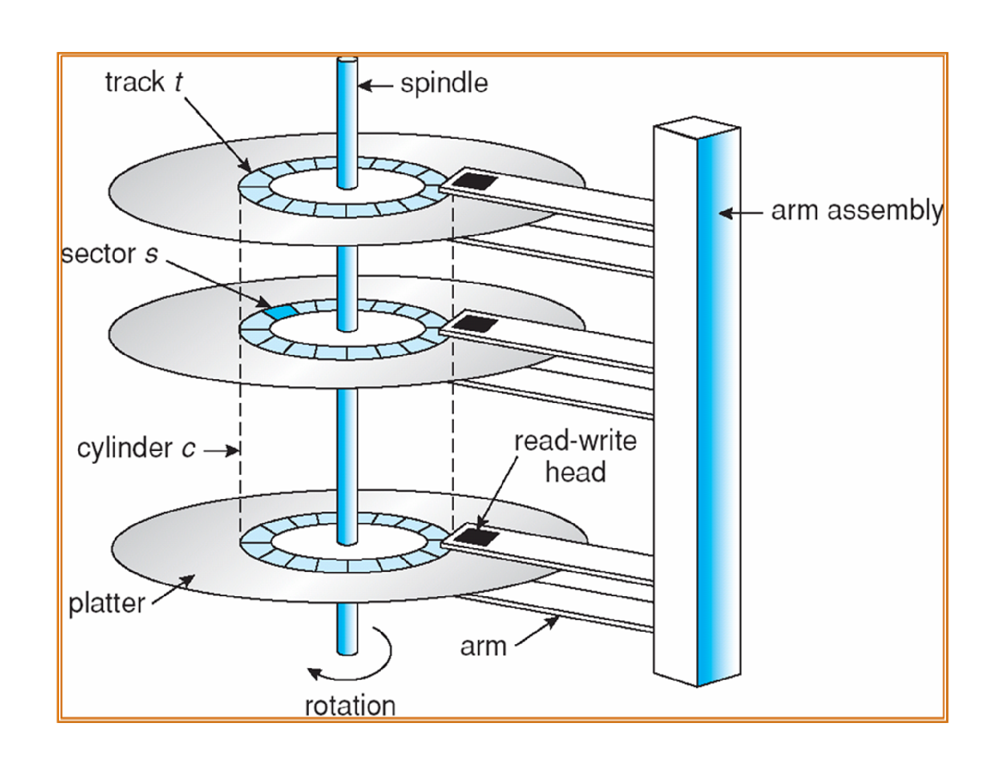
كامل مساحة هذه الأقراص . و يتحرك هذا السطح ذهاباً و إياباً ليتم التخزين على

صغيرة ، أية أجسام غريبة مهما كانت و الأقراص معاً داخل علبة محكمة الإغلاق لمنع دخول و توضع الرؤوس

. فأي جسم غريب قد يتسبب بتلف سطح القرص

➢ الأجزاء الإلكترونية :

إ لى مناطق ممغنطة على القرص ؛ ليتمكن) تحويل الإشارات الكهربائية (البيانات مهمته : و هو عبارة عن لوح إلكتروني بعد ذلك من استعادتها .



يتضح تركيب القرص المغناطيسي في هذا الشكل

جميع الأقراص الصلبة تعمل بنفس المبدأ و تختلف عن بعضها في جودة المكونات و سرعة عملها (سرعة المحرك) .

سرعة المحرك : مُعدّل دوران المحرك خلال الثانية .

تُزوِّد الأقراص المغناطيسية معظم التخزين الثانوي وتتراوح سرعة دورانها بين 70 - 250 دورة في الثانية وتختلف السرعة من قرص إلى آخر كالتالي :

- أقراص IPod : 4200
- أقراص الحاسوب المحمول : 4200,5400 أو 7200
- أقراص الحاسوب المكتبي : 7200
- أقراص الخادم : 10000 أو 15000

كلما ازدادت سرعة الدوران كان معدل استرجاع البيانات أعلى .

أداء محرك القرص الصلب:

هناك عدة مصطلحات تؤثر على أداء محرك القرص الصلب منها:

- **معدل نقل البيانات (Transfer rate):**

وهو مقدار(معدل تدفق) البيانات التي يستطيع نقلها بين القرص والذاكرة الرئيسية .

- **زمن القصد أو (وقت الطلب) (seek time):**

مقدار الوقت اللازم لتحريك الأذرع (أي الذي تأخذه رؤوس القراءة/الكتابة) للانتقال من مسار(track) إلى آخر .

- **تأخير الدوران أو (تأخير التدوير) (rotational latency):**

مقدار الوقت الذي تأخذه رؤوس القراءة/الكتابة لتجد قطاع (sector) يحتوي على موقع بيانات معين .

- **وقت الوصول (Access Time) :**

هو مقياس الوقت المتوسط الذي تأخذه رؤوس القراءة/الكتابة لتضع نفسها في مسار معين و لتجد قطاع يحتوي على موقع بيانات معين .

وقت الوصول = زمن القصد (seek time) + تأخيرالدوران (rotational latency)

- **وقت التوضيع أو (وقت التأشير) (positioning time):**

وهو الوقت اللازم لتأشير يد المحرك على المكان الصحيح .

خصائص القرص المغناطيسي :

إمكانية تخزين هائلة إذا تبدأ الأقراص (خاصة المرنة) بسعة تخزين تصل إلى أكثر من مليون حرف (بايت) .

وقد تصل سعة التخزين في بعض الأقراص الأخرى (الصلبة خاصة) إلى أكبر من جيجا بايت .

- سرعة وصول عالية وسرعة عالية في نقل البيانات.
- إمكانية تخزين كافة أنواع الملفات .
- إمكانية الوصول المباشر إلى المعلومات .
- إمكانية القراءة والكتابة في نفس الموقع (أي إمكانية تعديل البيانات فيها مواقعها إذا لزم الأمر).

مشاكل القرص الصلب:

- **ظاهرة تحطم الرأس (Head Crash)** :

تكون المسافة بين الرأس وسطح الطبق أقل من سماكة الإصبع – وكلما كان الرأس أقرب إلى الطبق , كلما كانت كثافة التخزين أكبر – حيث يكون الوضع العادي هو حركة الرؤوس فوق الأقراص .بمسافات صغيرة جداً بدون تلامس (تلامس رؤوس القراءة والكتابة غير مرغوب فيه مع سطح القرص) , وعند حدوث تلامس فإنه يتسبب في إتلاف سطح القرص أو الرأس نفسه.

وعند حدوث تلامس بسيط قد يمكن تلافي أثاره مع فقدان بعض البيانات في حين أن تلامس قوي قد يتلف القرص والرؤوس بالكامل. وقد يحدث هذا التلامس بسبب صدمة ميكانيكية أو اهتزازات شديدة أثناء التشغيل أو نقل مشغلات الأقراص بشكل خاطئ .

وقد تحدث نفس النتيجة بتلف سطح القرص بعد ضعف المادة المغناطيسية التي تغطي سطحه مع طول فترة الإستخدام ؛

وهو الذي قد يتسبب في تدمير البيانات الموجودة في المنطقة التي يلامس فيها الرأس الصفيحة (وفي حالة الصدمات العنيفة قد يحتك الرأس بسطح الصفيحة بأكمله) وهذه إحدى المشاكل الخطيرة التي تتعرض لها الأقراص الصلبة.

وعلى الرغم من إمكانية إعادة تهيئة القرص لتشغيله مرة أخرى إلا أنه يصاحب ذلك دائماً فقدان للبيانات وتتضمن المشاكل الميكانيكية، الخطيرة الأخرى: تلف الرؤوس ذاتها، والإخفاق المتعلق بتزويد الطاقة للدافع والمحركات الأخرى.

إلا أن مثل هذه الأعطال نادرة الحدوث، وإن حصلت، فليس هناك ما يمكن عمله حيالها. ويمكن، في بعض الحالات، إصلاح مشكلة اصطدام الرأس باستخدام برامج خدمية خاصة، أو لدى المراكز المتخصصة بصيانة الأقراص الصلبة، غير أن معظم البيانات، أو كلها، تكون بوضع لا يمكن معه إنقاذها .

يُذكر أنه توجد في بعض المحركات وسادة تمنع من حصول هذا الاصطدام تشبه في عملها تلك التي في السيارة .



تحطم للرأس حادّ على قرص صلب من نوع SCSI

نلاحظ في الشكل السابق وجود جسيمات غبار\على الطبق بسبب تعرض القرص للهواء الخارجي .

تبدو جسيمة الغبار على الطبق وكأنها جبل بالنسبة لرأس القراءة / الكتابة , ويمكن أن تسبب أضراراً كارثية للمحرك , وللحفاظ على الهواء نظيفاً داخل المحرك يوجد في كل محركات القرص الصلب فتحات دقيقة تحوي مرشحات , وذلك ليكون هناك توازن في الضغط بين داخل وخارج المحرك .

➢ **الطاقة :**

مشاكل القرص الصلب كثيرة حيث أنه علاوة على حدوث تحطم في الرأس فإنه يتطلب طاقة تقدر بـــ120 فولت .

يتم ربط المحرك بالحاسوب عن طريق ممر البيانات الخارجي (EDB) . ويتم ربط الأقراص الصلبة مع لوحة الأم للحاسب بكيبل خاص .

واجهات محرك القرص الصلب عديدة منها

EIDE,ATA,SATA,Fireware,USB,Fiber Channel,SCSI:

المتحكم المُضيف (Host Controller) في الحاسوب : يستخدم ممر البيانات الخارجي للتحدث مع متحكم القرص الموجود داخل المحرك .

الأقراص الصلبة ممكن أن تكون قابلة للفصل أي أنها ممكن أن تكون خارجية .

متحكم القرص :

هو دارة الدعم التي تعمل كصلة وصل بين محرك القرص الصلب وممر البيانات الخارجي .

أنواع الأقراص :



IDE ATA Internal Hard Disk ➢ :

(Integrated Drive Electronics): من IDE يأتي تعريف اسم

. Parallel ATA و أحيانا يطلق عليه اسم PATA وهو ما يسمى

وهو يعتبر من الأنواع الشهيرة في الإستخدام لسنوات طويلة ، وهو من الأقراص ذات الجيل الأول

في استخدام الحواسيب المترلية والمكتبية . ويأتي عادة بسعات تتراوح بين 4 GB و GB 200.

والسعات في تزايد مع سباق الشركات لذلك .

ولكن في الآونة الأخيرة تم التوجه إلى استخدام نوع آخر من أنواع الأقراص الصلبة والذي يمتاز بسرعة أعلى منه. وتسمى أقراص SATA.

SATA Internal Hard Disk ➢ :

SATA جاء تعريف القرص الصلب

(من Serial Advanced Technology Attachment.)

وهذا النوع من الأقراص الصلبة ، كما ذكرنا سابقاً يمتاز بسرعة أعلى من القرص الصلب .

مما يجعل تشغيل البرامج والوصول الى البيانات أفضل وأسرع من ذي قبل .

. IDE وتأتي تلك الأقراص الصلبة مع أجهزة الحاسوب الجديدة الصنع

SCSI Internal + External Hard Disk ➢ :

SCSI جاء تعريف القرص الصلب

(من (Small Computer System Interface .

وهو نوع خاص من الأقراص الصلبة التيُ تستخدم لأجهزة الخوادم الصغيرة الحجم والعملاقة ,

. وبشكل تلقائي يتم تثبيت تلك الأنواع من الأقراص الصلبة في جميع حسب احتياج الشركات لعمليات التخزين

الخوادم لما لها من سرعة محددة تعتبر أسرع من النوعين السابقين ، وعادة تأتي بأحجام مختلفة .

ربط (وصل) القرص بلوحة الأم :

كل قرص صلب لابد من توصيله باللوحة الأم حتى يمكن نقل المعلومات من وإلى القرص ، وحتى نفعل ذلك

. "لابد من وجود جهاز ما , يوصّل هذين الشيئين وهذا ما يسمى : " البينية و كما ذكرنا سابقاً الأقراص الصلبة 3 أنواع .

كل قرص صلب متوافق مع نوع معين من البينيات ولا يمكنه العمل مع سواها .

، لتوصيله في اللوحة الأم .) مدخل الخاص به(فكل نوع يحتاج إلى كيبل خاص به

- EIDE ونأتي على ذكر النوع الأول

وهو خاص بالنوع الأول آنف الذكر .

ويمكن تسميتها اختصاراً بـــ " IDE " وترجمة الاسم هي : " السواقة ذات الإلكترونيات المضمنة

والمحسنة" . و معنى الاسم أن الإلكترونيات اللازمة لتشغيل القرص موجودة بداخله (لوحة التحكم) وليس خارجه .

وهي بلا منافس الأكثر شيوعاً بين المستخدمين في الوقت الحالي .

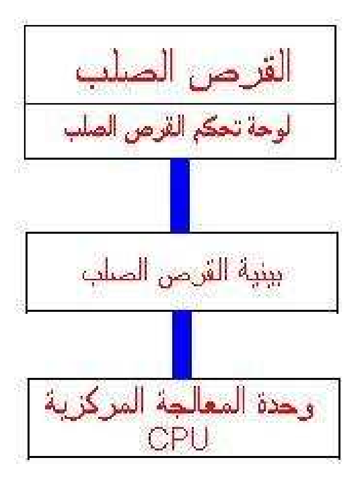
وفي هذا النوع من الأقراص الصلبة يوجد بينية (في الماضي كان بطاقة توسعة أما الآن فهي مدمجة في جميع

اللوحات الأم) لها مشبك خاص يدعى مشبك IDE .

(أنظر الشكل التالي) ويوصل كيبل خاص من القرص الصلب إلى مشبك IDE

و تستقبل بينية IDE الطلبات من المعالج وتقوم بالتفاهم مع لوحة التحكم الخاصة بالقرص لجلب البيانات المطلوبة.

. وتستخدم هذه الموصلات نفسها لوصل مشغلات الأسطوانات المدمجة



تتسع بينية EIDE الواحدة إلى أربعة أجهزة IDE موزعة على قناتين : أولية وثانوية بواقع جهازين لكل قناة . تتقبل بينية IDE أية أجهزة متوافقة مع مواصفات IDE سواء أكانت أقراص صلبة أو أي أجهزة أخرى مثل محركات الأقراص المدمجة CD أو DVD أو أجهزة التخزين الاحتياطي الأخرى .

- SATA أما النوع الثاني الحديث

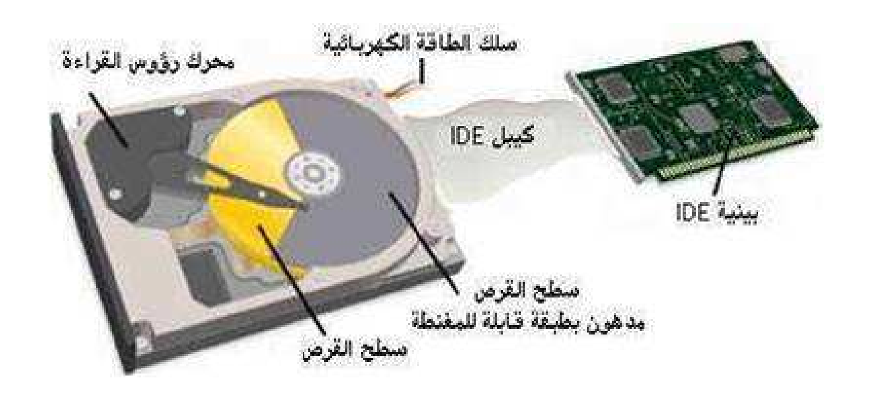
و الذي بات استخدامه الآن أكثر من النوع الأول .

وهو الذي يختص بالقرص الصلب هو كيبل من نوع آخر يأتي بلون أحمر وأصغر حجما ولا تلزمه صعوبة في التركيب . الأول ، مقارنة بالنوع

SCSI والنوع الأخير، و الذي يطلق عليه اسم ➢

وهي أسرع من الأولى و لكنها أغلى بكثير ، وتعتبر أفضل ميزة فيها سرعتها الكبيرة في التعامل مع طلبات كثيرة في نفس الوقت لذا فهي غالباً لا تستخدم إلا في الأجهزة الخادمة .
تعمل أجهزة سكزي بطريقة مختلفة عن الـــ IDE فهي عبارة عن مجموعة من الأجهزة (أقراص صلبة أو أجهزة تخزين أخرى مثلاً) مربوطة مع بعضها بناقل خاص يمكنها – بخلاف IDE – من تبادل البيانات مع بعضها بدون تدخل المعالج المركزي ، فلو أردنا مثلاً نسخ ملف من قرصين صلبين من نوع سكزي فسوف يتم ذلك بدون إشغال المعالج ، فيمكننا إذاً أن نقول أن هذه الأجهزة مستقلة بذاتها .

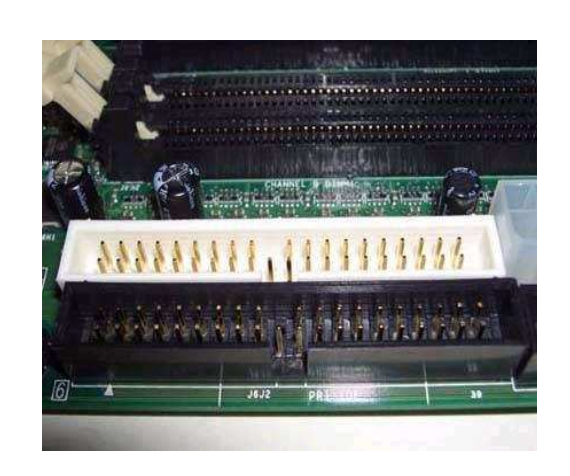
وكما هو الحال مع IDE تتطلب هذه البينية مشبك سكزي ولكن بخلاف IDE فإن هذا المشبك لا يوجد غالباً على اللوحة الأم بسبب ارتفاع تكلفته وندرة استخدامه لذا فلا بد من تركيبه بواسطة بطاقة توسعة تركب على اللوحة الأم وتوصل بها أجهزة سكزي . وتعتبر أجهزة سكزي سريعة جداً ولكنها بالمقابل صعبة التركيب وتعاني من مشاكل التوافقية في بعض الظروف .



. الصلب من خلاله وهو يحتاج الى كرت إضافي يتم توصيله باللوحة الأم لتوصيل القرص



طريقة ربط الكيبل مع القرص



طريقة تركيب الكيبل مع لوحة الام

12-1-2 الشريط المغناطيسي :

هو أحد وسائط التخزين الثانوية المبتكرة ويحمل كميات دائمة و كبيرة من البيانات.

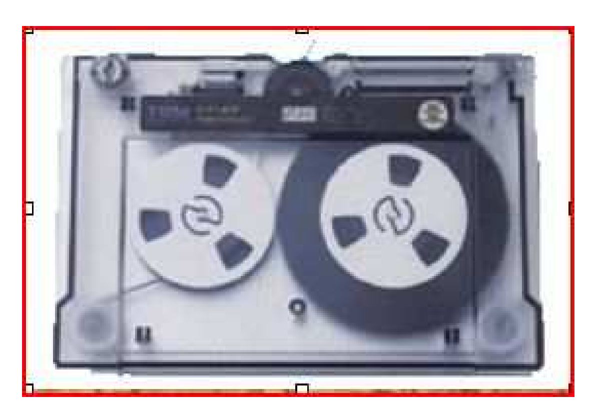
زمن الوصول (**access time**):

يكون بطيء مقارنة بالذاكرة الرئيسية و القرص المغناطيسي .

الوصول العشوائي (**random access**):

الوصول العشوائي للشريط المغناطيسي أقل بألف مرة من الوصول العشوائي للقرص المغناطيسي , هذا ما يجعل الأشرطة ليست مفيدة جداً في التخزين الثانوي .

ولذلك : يُستخدم الشريط المغناطيسي بشكل رئيسي للإسناد ولتخزين البيانات التي تستخدم نادراً وكذلك يُستخدم كوسط لنقل المعلومات بين الأنظمة .



صورة للشريط المغناطيسي

مشاركة الطالبات:

ريهام حافظ– منيرة عبدالله بن هريس–نهله محمد– الجوهرة الصحن– هند المطيري

المراجع:

- **جريدة القبس**
- **(Operating System Concepts) كتاب**
- **(CompTIA A + Certification) كتاب**
- **WWW.IFOREX.COM**
- **http://www.sanabes.com/**
- **http://www.dcs.ed.ac.uk/home/stg/pub/D/disk.html**

2-12 تركيب القرص:

إن لم يكن أكثر , (Hard Disk) تحتوي معظم أجهزة الكمبيوتر اليوم علي قرص صلب و غيرها (Servers) بل إن العديد من الحاسبات الكبيرة مثل أجهزة الخادمات تحتوي على المئات من الأقراص الصلبة وبأحجام كبيرة ، لتشغيل الجهاز . ولكن لا يعتبر وجود القرص الصلب ضرورة مُلِحّة يتمثل الدافع الرئيسي وراء استخدام هذه الأقراص الصلبة في شئ واحد وهو أنها تستطيع الاحتفاظ بالكثير من البيانات بعد أن تفصل الكهرباء عن الحاسب .

حيث يستطيع القرص الصلب أن يخزن البيانات الرقمية على هيئة مغناطيسية تدوم طويلاً التعامل مع القرص الصلب (Formatting the HDD) تهيئة القرص الصلب لكي نستطيع استخدام القرص الصلب يجب أن نقوم بتهيئته أولا .

هناك نوعان من التهيئة :

- (Low Level Formatting) تهيئة المستوي المنخفض

التهيئة الفيزيائية . (Physical Formatting)

- (High Level Formatting) تهيئة المستوي العالى

التهيئة المنطقية . (Logical Formatting)

مشاركة الطالبة:

نهله محمد.

المراجع:

- **WWW.IFOREX.COM**

12-3 ارتباط القرص:

تصل الحاسبات إلى أجهزة التخزين بطريقتين.

الطريقة الأ ولى عبر منافذ الإدخال والإخراج (I/O ports) ، (أو مخزن ربط المُضيّف)

وهذه الطريقة مشهورة في الأنظمة الصغيرة . أما الطريقة الأخرى فهي تُدعى (مخزن ارتباط الشبكة) وتتم عبر المُضيّف البعيد في أنظمة الملفات الموّزعة .

12-3-1 **مخزن ارتباط–الشبكة (NAS)** :

(.Network attachment storage)

هو تعبير يستخدم لوصف نظام التخزين المتكامل المصمم لربط بيانات الشبكة.

أنظمة مخزن ارتباط– الشبكة (**NAS protocols**):

تدعم خادمات (NAS)(servers) نظام ملفات الشبكة (NFS) ونظام ملفات الانترنت المشتركة (CIFS) ،وقد تدعم أيضا نظام إرسال الملفات (FTP) .

اتصالات مخزن ارتباط– الشبكة (**NAS connections**) :

خادم (NAS) يرتبط غالبا بالشبكة عن طريق الإيثرنت .

2-3-12 شبكة منطقة–التخزين (SAN) :

(Storage area network)

هي شبكة فرعية لأجهزة التخزين المشتركة منفصلة عن (WAN)و(LAN)

وتستخدم لربط كل مصادر التخزين بالخادمات المختلفة (servers)

تعريف الــ SAN:

شبكة تخزين البياناتSAN هي معمارية لربط الحاسبات النائية وأجهزة التخزين مثل مصفوفة الأقراص (disk arrays)، (Tape libraries) و (optical jukeboxes) إلى الــخادمات(Servers) بطريقة تبدو إلى نظام التشغيل كأنها مرفقة محلياً بالجهاز (Locally attached).

بالرغم من أن التكلفة والتعقيد بدأت بالتناقص – في عام 2007، إلا أنه ما زالت SAN غير مألوفة خارج المؤسسات الكبيرة (Enterprise).

مزايا الــ SAN:

اليوم تأتي في نوعين قد تكون رئيسية: قناة الألياف الضوئية FC(Fiber Channel) و iSCSI او IP-based SANs.

قناة الالياف هي اكثر نوع معروف من SAN، ولكن على مدى العامين الماضيين، iSCSI قد بدأت تنتشرفي السوق بطريقه كبيرة، ويرجع ذلك الى حسن الاداء وقلة التكلفة مقارنة بقناة الألياف.

لكن الأفضل الجمع بين كل من DAS(Direct Attached Storage) و NAS.

على سبيل المثال، مع التنفيذ السليم، يمكن الحصول على redundant شبكة تخزين يمكن توسيعها الى مئات التيرابايت أو الى الـــ NAS ؛

لكن لن نحصل على مستوى الوصول الى البيانات كما يحقق ذلك DAS (مثال الــ DAS هو الــ Hard Disk).

كما يمكن الوصول الى البيانات في سرعة معقولة، مما يجعل SANs جيد حتى للعمليات التي تتطلب الحصول على معدل كبير للقرص مثال على ذلك قواعد البيانات والايميل سيرفر.

مع SAN، عليك ايضا ان تحصل على مركزية التخزين حتى مع بعض التطبيقات، يمكنك عمل الـــخادمات (Servers) بدون أي خدمة تخزين داخلي وتحتاج الى ان جميع النظم تعمل الإقلاع (boot)مباشرة من SAN (تدعمها قناة الألياف فقط).

سلبيات SAN:

مع كل هذه النقاط الكبيرة، اذن ماهي سلبيات SAN؟

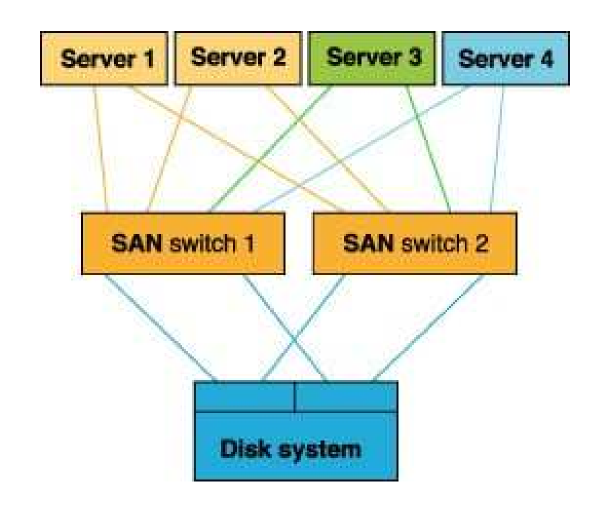
هناك اثنان من العوائق الرئيسية لـــ SAN: التكلفه والتعقيد، خصوصاً عندما يتعلق الامر بالألياف (Fiber Channel).

ألياف SAN تكلف من K$50-60 لمجرد تيرابايت من التخزين.

من ناحية اخرى، SAN الذي يعمل على iSCSI يبدأ بسعر K$20-30 (عشرين الى ثلاثين الف دولار)،

لكن لاتصل الى مستويات الاداء التي تقدمها الالياف وحتى المسافة التي تصل الى 6 ميل.

الفرق في السعر يرجع إلى ان معظم iSCSI قادرة على الاستفادة من اجهزة الجيجابيت إيثرنت، في حين تتطلب الالياف أجهزة متخصصه، ومعدات مكلفه.

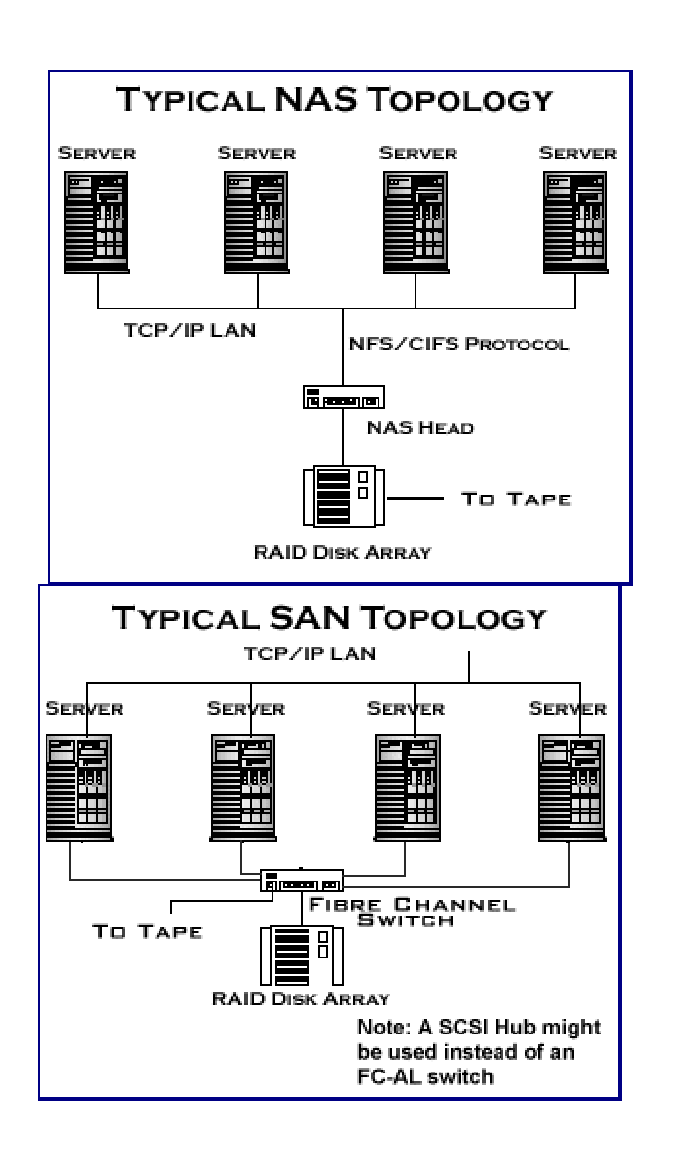


SAN

12-3-3 مقارنة بين مخزن ارتباط–الشبكة (NAS) و شبكة منطقة–التخزين (SAN) :

من النظرة الأولى نجد أن (NAS)و(SAN) متماثلين ولكن في الحقيقة هما مختلفين ، وهنا بعض الاختلافات :

NAS and SAN



SAN	NAS
- نظام الملفات يدار بواسطة الخادم .	- نظام الملفات يدار بواسطة وحدة (NAS).
- مشاركة الملفات نظام تشغيل تابع وغير موجود في العديد من أنظمة التشغيل.	- يسمح بمشاركة المعلومات بين أنظمة التشغيل المختلفة مثل يونيكس و NT .
- فقط أجهزة (server) مع (SCSI fiber channel) تستطيع الاتصال بـــ (SAN) ، و	- تقريبا أي آلة تستطيع أن ترتبط بالشبكة المحلية (LAN) أو ترتبط بالشبكة المحلية عن طريق

الشبكة الواسعة (WAN).	(fiber channel) تأخذ 10 كم في أحسن الأحوال كحد أقصى .

مشاركة الطالبات:

إيمان الزهراني– عهود

المراجع:

- http://en.wikipedia.org/wiki/Network-attached_storage
- http://www.nas-san.com/differ.html

4–12 جدولة القرص :

في الأنظمة متعددة المهام (متعددة البرامج) هناك عدة عمليات مختلفة تريد استعمال مصادر النظام في وقت واحد. لذلك يحتاج مشغل الأقراص إلى آلية لحل هذا التراع ، ومشاركة المصدر بين العمليات بإنصاف .

إذا كان كلٍ من مشغل القرص ومتحكم القرص (controler) متاح فإن المهمة ستُنَفذ مباشرة ، أما إذا كان أحدهما مشغولاً بتنفيذ مهمة ما ؛ فإن أي مهمة جديدة سوف تضاف إلى قائمة المهام المُعلّقة (queue) .

وعندما تكتمل المهمة قيد التنفيذ فإن نظام التشغيل هو المسئول عن اختيار المهمة التالية من قائمة المهام المُعلقة ليتم تنفيذها .

وهذا يتم بناءً على خوارزميات جدولة معينة يستخدمها القرص (scheduling policies) .

من أهم مسئوليات نظم التشغيل إستخدام الأجهزة بكفاءه وفي حالة القرص الصلب لابد من الإهتمام بعدة معايير منها : تسريع وقت الوصول , وعرض القرص أيضاً .

- يندرج تحت هذا المعيار مفهومين (acess time) : وقت الوصول

وهو الوقت الازم لتحريك راس القرص إلى المكان المطلوب. (seek time)المفهوم الاول: وقت الطلب

وهو الوقت المنتظر لتدوير القرص لوضع الرأس عليه.(rotational latency) المفهوم الثاني: تاخير التدويير

- (disk bandwidth)عرض القرص

وهو العدد الإجمالي للبايت المنقولة مقسومة على الوقت من طلب الخدمة وحتى الانتهاء منها.

الطاقة الانتاجية throughput (متوسط عدد الطلبات في الوحدة الواحدة)

و وقت الرد response time (متوسط الوقت بين وصول الطلب وبداية الطلب)

انتقالات الرأس بين مسارات الإسطوانة يتطلب وقت طويل وحتى نقلل من طلبات الإدخال والإخراج نستخدم الجدولة (scheduling) لكي نقلل من حركة الرأس.

من الناحية الأخرى فإن تقليل حركة الرأس head تُرضي الطلبات القريبة من الموقع أما الطلبات البعيدة فإنها قد تنتظر لوقت طويل .

وفهمنا للمعايير السابقة يساعدنا في تقييم أداء خوارزميات جدولة القرص ومعرفة الأفضل منها .

- **أولاً : جدولة القادم أولاً يُخدم أولاً (FCFS) :**

(FCFS) هو اختصار (First Come First Serve) ويعني (الأول في الوصول يخدم أولاً)

حيث تتم الخدمة على حسب ترتيب وقت الوصول ومن ثم التحرك ذهاباً وإياباً عبر سطح القرص للوصول إلى المكان المطلوب , حيث يتحرك تقريبا بطريقة عشوائية على سطح القرص. و حين يأتي طلب جديد ينتظر دوره .

لكن هذا قد يستهلك الكثير من الوقت مما يؤدي إلى تقليل سرعة تقديم الخدمات .

وهو أبسط شكل لجدولة القرص .

مثال:

لوكانت قائمة المهام المُعلّقة (queue) أو (صف مهام الإدخال والإخراج) لأجزاء من الإسطوانات

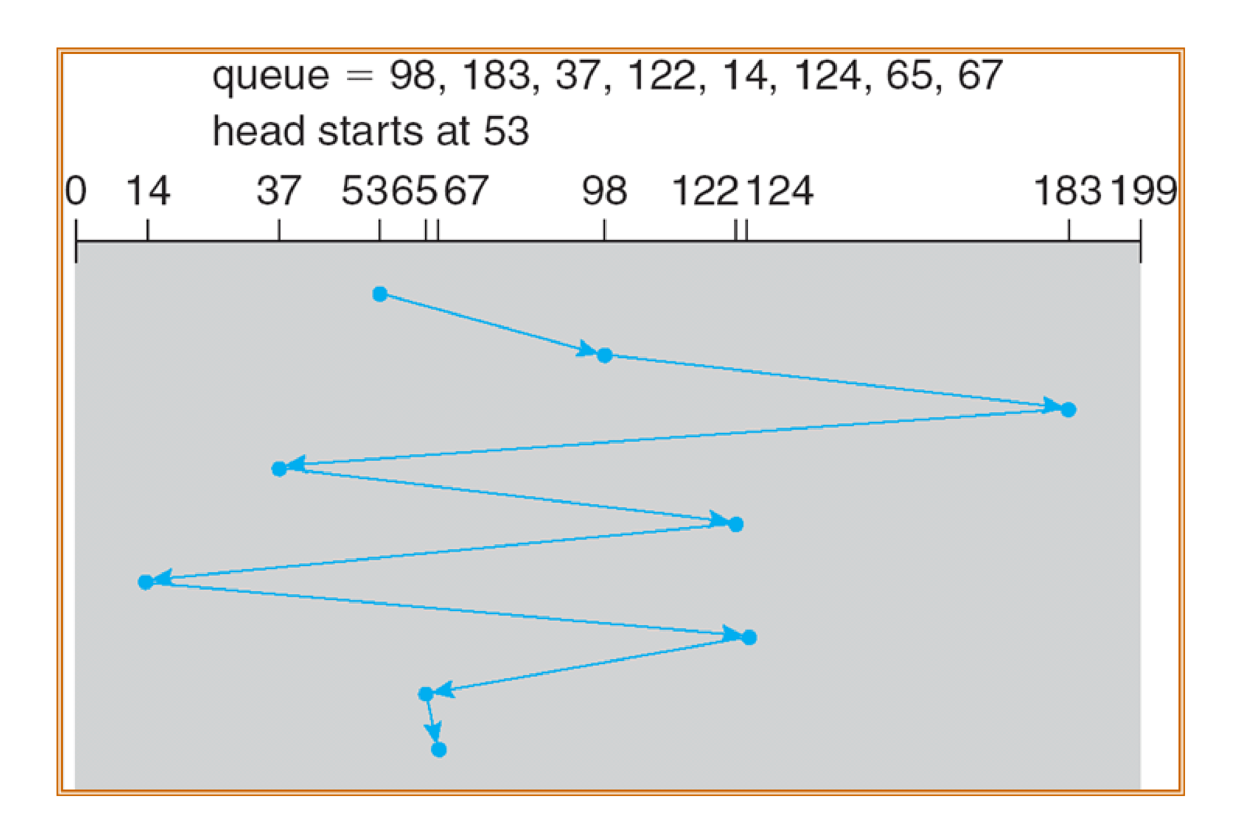
كالتالي (بالترتيب) :

76,65,124,14,122,37,183,98

وكان رأس القراءة والكتابة عند الأسطوانة رقم53 ؛ فإنه في البداية سينتقل من 53 إلى 98 بحيث يمر خلال 45 أسطوانة

(98 - 53=45) ثم ينتقل من 98الى 183 ليمر خلال 85 أسطوانة وهكذا حتى يصل إلى الأسطوانة رقم 67 .

نلاحظ التنفيذ سيكون بالترتيب كما في الصورة



سيتحرك الرأس من 53 إلى 98 ثم إلى 183 ثم يعود إلى 37 ثم 122

. 67   ثم يعود إلى 14 ثم 124 ثم يعود إلى 65 وأخيراً إلى

ولكي نحسب مجمل تحرك الرأس نقوم بجمع القيم المطلقة للفرق بين مكان وجود الرأس والمكان الذاهب إليه كما يلي :

إجمالي تحرك الرأس :

tota head movment = | 53 - 98 | + |98 - 183 | + |183 - 37 | + |37 - 122 | + |122 - 14 | + | 14 - 124 |

+ | 124 - 65 | + | 65 - 67 |

= 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2

= 640 cylinders

وبذلك يكون رأس القراءة والكتابة قد انتقل خلال 640 أسطوانة لإنهاء مهام الإدخال والإخراج للقرص .

المشكلة في هذا النوع من الجدولة : عندما انتقل رأس القراءة والكتابة من الأسطوانة 37 إلى الأسطوانة 122 ثم عاد إلى الأسطوانة 14ثم إلى الأسطوانة 124 . وهذا ما يعرف بـــ (wild jump).

حيث ستكون الفعالية أفضل لو أنه أنهى العمل من 37 و 14 تباعاً ثم انتقل للعمل على 122 و 124 لأن هذا سيقلل من عدد الأسطوانات التي سيمر عليها .

خصائص خوارزمية FCFS :

- نؤدي العمليات على حسب ترتيبها المتطلب .
- لا يوجد إعادة ترتيب في عمل طابور العمليات .
- لا يوجد تجويع لإحدى العمليات.
- سيئة من ناحية معيار جودة الأداء تقلل زمن الاستجابة .

- **ثانياً : جدولة الأقصر يخدم أولاً** (الجدولة بحسب زمن البحث) (**SSTF**) :

(SSTF) هو اختصار(shortest-seek-time-first) يعني (الأقصرفي زمن البحث أولاً)

فهذا النوع أكثر فعالية من النوع السابق . يعتمد هذا النوع على خدمة أو تنفيذ المهام الخاصة بالأسطوانات الأقرب إلى موضع رأس القراءة والكتابة الحالي قبل الإنتقال إلى الأسطوانات البعيدة . ويتم ذلك بختيار المهام ذات زمن البحث الأقصر .

حيث أنه من المنطقي خدمة جميع الطلبات القريبة من الرأس قبل توجه الرأس بعيداً لخدمة الطلبات الأخرى .

وهو أساس هذه الخوارزمية. و هذا الإفتراض يحل المشكلة الظاهرة في الخوارزمية السابقة)حيث تقوم هذه الخوارزمية بإختيار الطلب صاحب الفرق الأقل من مكان الرأس الحالي(أي أنها تعالج الأوامر على حسب قصرها , بصرف النظر عن الاتجاه , فا لحركة على سطح القرص عشوائية لكن الوقت المقضي في الحركة أقل , وهذه الآلية أفضل من الخوارزمية السابقة . وبالرغم من أنها أفضل من الخوارزمية السابقة إلا أنها ليست الأفضل .

مثال:

إذا كان رأس القراءة والكتابة عند الأسطوانة رقم53 ، ستكون أقرب أسطوانة من الموقع الحالي للرأس

هي الأسطوانة 65 لأن | 53 - 65 | = 12,

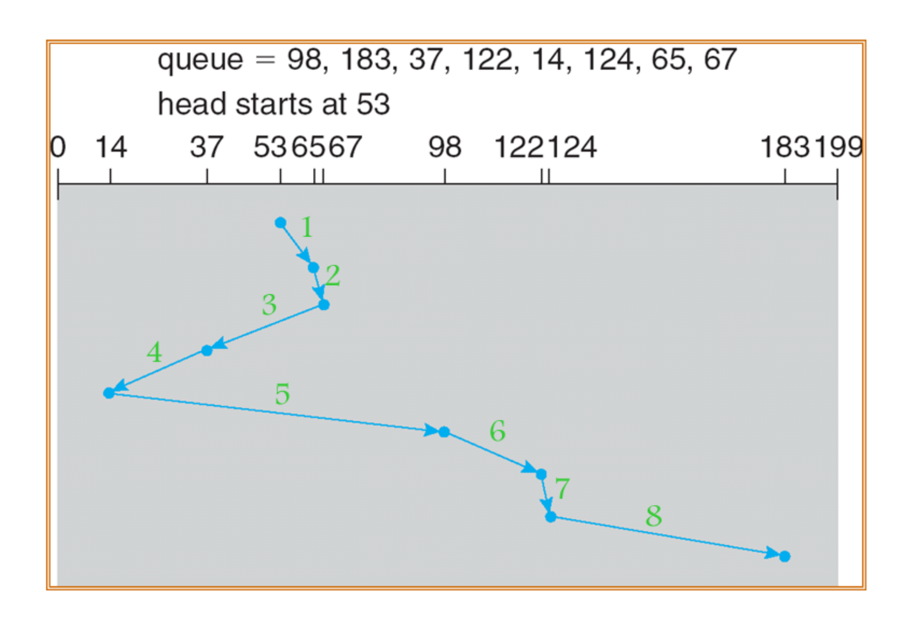
بعد ذلك تكون النقطة الأقرب هي 67

. ثم نكمل الترتيب :31 = | 98 - 67| و : 30 = | 37 - 67| ثم 37

فـــــ 37 أقرب من 98 لأن

ثم 14 ثم 98 ثم 122 ثم 124 ثم 183 .

الصورة التالية توضح هذه الخوارزمية



ولكي نحسب مجمل تحرك الرأس :

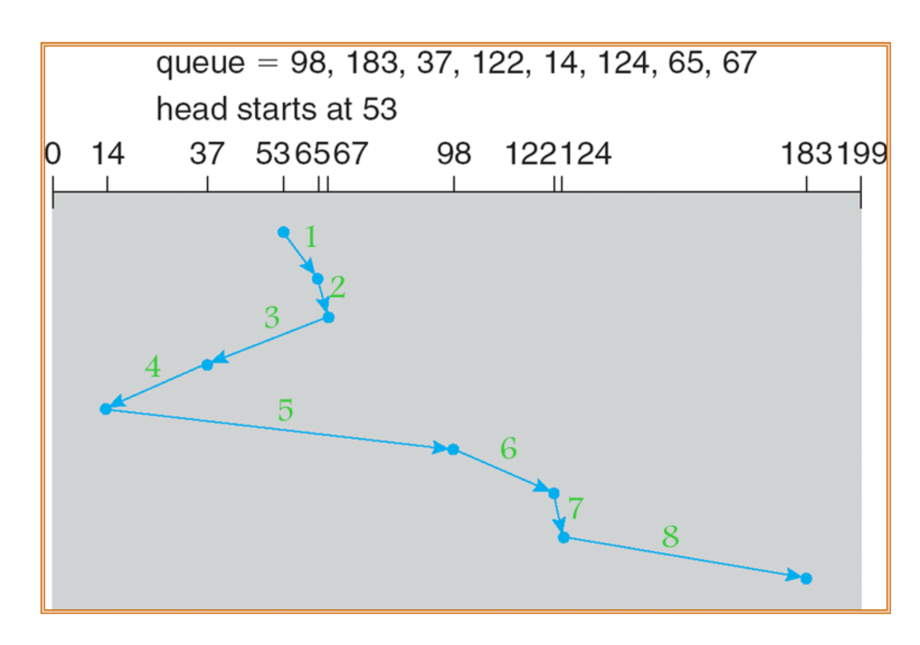
tota head movment = | 53 - 65 | + |65 - 67 | + |67 - 37 | + |37 - 14 |
+ |14 - 98 | + | 98 - 122 | + | 122 - 124 | + | 124 - 183 |

= 12 + 2 + 30 + 23 + 84 + 24 + 2 + 59

= 236 cylinders

وبذلك يكون إجمالي الأسطوانات التي انتقل الرأس خلالها هو 236 وهو ما يقارب ثلث عدد الأسطوانات التي مر خلالها الرأس في الخوارزمية الأولى السابقة .

وبالرغم من أن 37 ليست الأقرب لـــ 53 إلا أننا نجد في هذه الخوارزمية ستكون الفاعلية أفضل لو انتقل الرأس إلى 37 أولاً ثم إلى 14 ثم 65 ثم 67 ثم 98 ثم 122 ثم 124 ثم 183 حيث سيكون إجمالي الأسطوانات التي سيمر عليها 208 وهو أقل .



في هذه الخوارزمية هناك احتمالية أن أحد الطلبات قد يُؤَخر فترة طويلـــة وذلك إذا وصلت بعده طلبات كثيرة أقصر منه .

وهذا يعني أن هذه الخوارزمية قد تسبب التجويع (starvation).

كذلك هناك مشكلة أخرى في هذا النوع من الجدولة : وهي وجود بطء أثناء التحول في الاتجاهات .

➢ ثالثاً : جدولة الفحص (المسح)– (**SCAN**) :

في هذه الجدولة يبدأ ذراع القرص الصلب (arm) من أحد أطراف القرص ثم يتحرك نحو النهاية الأخرى ,خادماً الطلبات التي يمر بها عندما يصل كل اسطوانة ، وذلك حتى يصل إلى النهاية الأخرى من القرص.

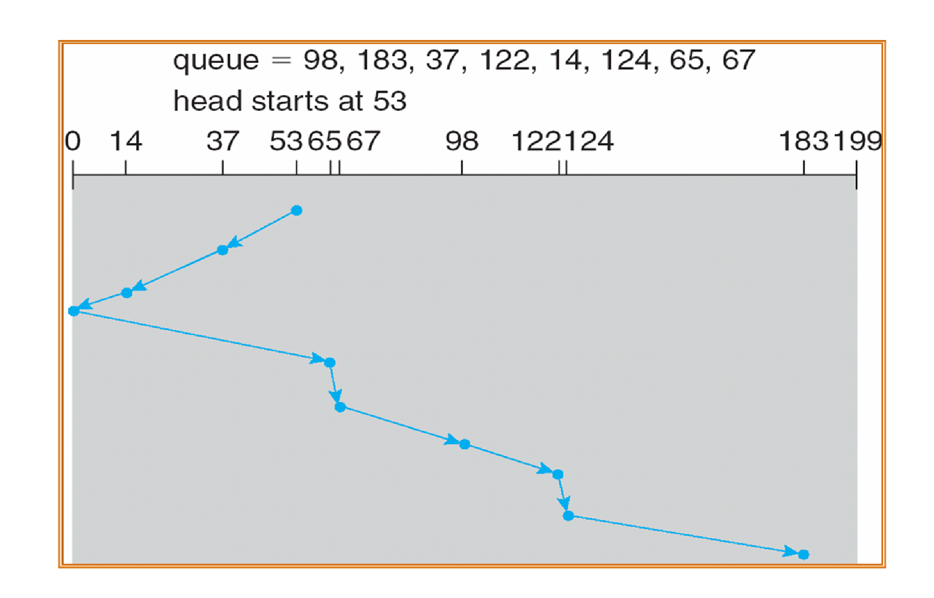
وعندما يصل تلك النقطة فإن اتجاه الحركة يُعكس ، ويُكمل تنفيذ الطلبات بالإتجاه الآخر.

ويكون الفحص (المسح) بشكل مستمر ذهاباً وإياباً عبر القرص الصلب.

الحركة هنا تكون أقل من (القادم أولاً يخدم أولا) و أكثر عدلاً من (الأقصر يخدم أولا).

هذه الجدولة تُسمى أحيانا : **خوارزمية المصعد** (elevator algorithm)

وذلك لأن ذراع القرص الصلب يتصرف مثل المصعد تماماً.



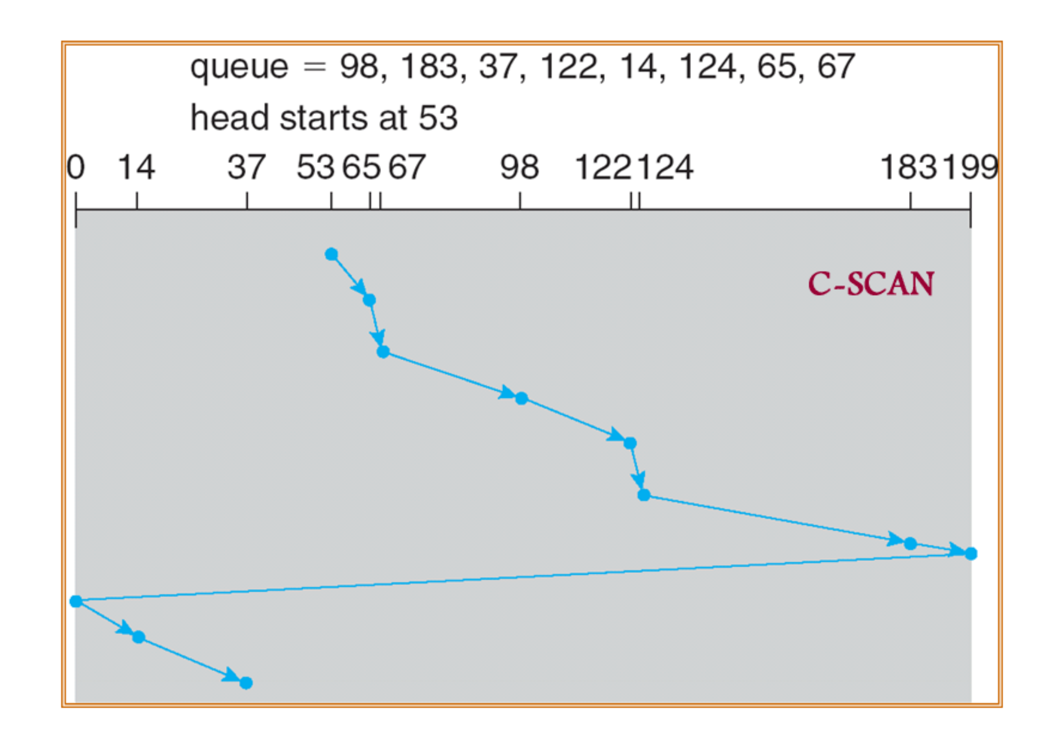
➢ رابعاً : جدولة الفحص الدائري (المسح الدائري)– (**C – SCAN**) :

هذه الجدولة تختلف عن تصميم خوارزمية الفحص (المسح) السابقة حيث أنها ُتقدم زيّ يحوي رسمية أكثر .

فهي تشبه خوارزمية المسح في أنها تبدأ من أحد طرفي القرص الصلب إلى الطرف الآخر،

خادمةً بذلك الطلبات التي تُواجهها في طريقها ؛ لكن عندما تصل إلى الطرف الآخر فإنها تعود على الفور

إلى بداية القرص الصلب ، دون أن تخدم الطلبات التي في طريق العودة , ومن ثم يبدأ بأخذ الطلبات .



فهي تعامل الإسطوانات وكأنها قائمة دائرية (circular list) تلتف حولها من آخر اسطوانة إلى أول اسطوانة.

ولكي نحسب مجمل تحرك الرأس :

tota head movment = | 53 - 65 | + |65 - 67 | + |67 - 98 | + | 98 - 122 | + |122 - 124 | + | 124 - 183 | + | 183 - 199 | + | 199 - 0 |+ | 0 - 14 | + | 0 - 37 |

= 12 + 2 + 31 + 24 + 2 + 59 + 16 + 199 + 14 + 37

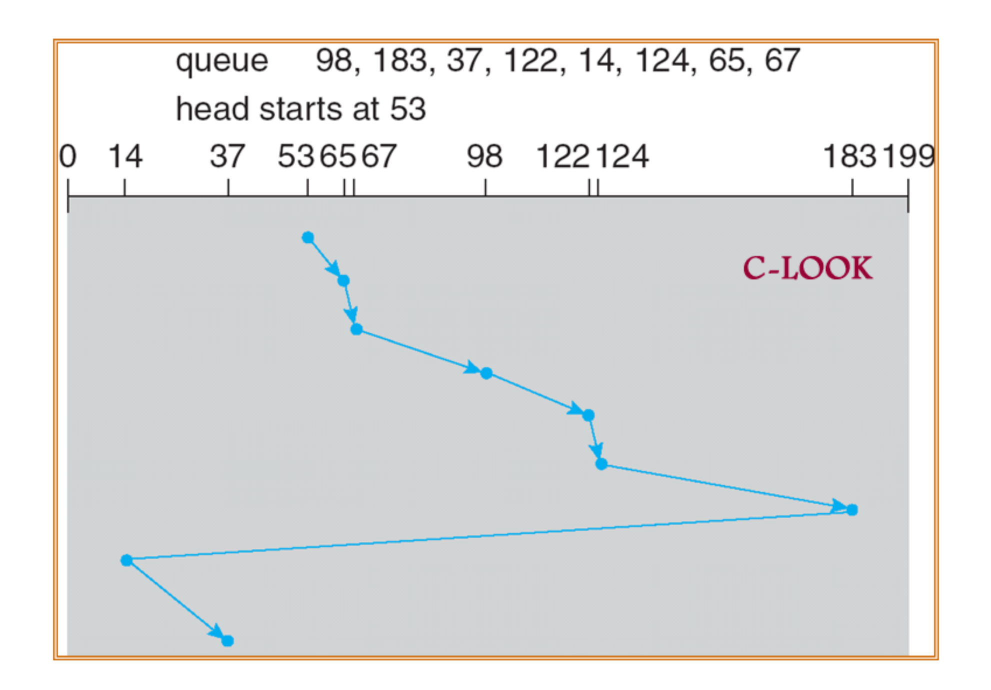
= 396 cylinders

➢ خامساً : جدولة النظرة (LOOK) :

ذكرنا في الجدولتان السابقتان (c-scan) ، (scan) ، أن ذراع القرص الصلب يتحرك إلى نهاية القرص الصلب ؛ لكن عملياً لايوجد خوارزمية تبرمج هذه الطريقة.

هذه الخوارزمية تشبه خوارزمية الفحص (المسح) إلا أن الذراع يتحرك إلى آخر طلب في القرص في كل اتجاه وليس إلى نهاية القرص الصلب . ثم يعكس اتجاهه على الفور ، دون أن يذهب إلى نهاية القرص .

وذلك لأنها تبحث عن الطلب قبل الاستمرار في الحركة في الاتجاه المعطى .



الإصدارات من (c-scan) ، (scan) التي تتبع هذه الطريقة تُدعى : LOOK و C-LOOK

ولكي نحسب مجمل تحرك الرأس :

tota head movment = | 53 - 65 | + |65 - 67 | + |67 - 98 | + | 98 - 122 | + |122 - 124 | + | 124 - 183 | + | 183 - 14 | + | 14 - 37 |

= 12 + 2 + 31 + 24 + 2 + 59 + 169 + 23

= 322 cylinders

مشاركة الطالبات:

الجوهرة الربيعه – ريهام حافظ – ليلى بنت علي البيشي – ماحدة بن طالب – أمل الصبيح – عالية المصيريعي – لمياء السدحان – **هنادي الحزيمي – سديم الحبيب**

المراجع:

- Operating Systems Concepts book
- http://www.dcs.ed.ac.uk/home/stg/pub/D/disk.html
- http://www2.cs.uregina.ca/~hamilton/courses/330/notes/io/node7.html
- http://www2.cs.uregina.ca/~hamilton/courses/330/notes/io/node7.html
- هذا رابط لموقع يقوم برسم اتجاه الحركة مع الحسابات :
- http://gaia.ecs.csus.edu/~zhangd/oscal/DiskApplet.html

6-12 Swap-Space Management:(إدارة فضاء التبديل)

في الفصل الثامن , (swapping)في السابق تم ذكر عملية التبديل

. حيث تتم هذه العملية مؤقتا بين القرص الصلب والذاكرة ليتم تبديل العمليات لكن في ظل أنظمة التشغيل الحديثة ، قليل منها من يتبع تلك الطريقة في تنفيذ عمليات التبديل.

(Virtual Memory techniques)(حيث أنه الآن ، ومع وجود تقنيات الذاكرة الافتراضية

(processes).) وليس العمليات (pages فان الأنظمة تقوم باستبدال الصفحات

1-6-12 Swap-Space Use):(استخدام فضاء التبديل

التبديل أو (المبادلة):

تعرفنا في فصول سابقة على مفهوم التبديل والذي كان ينص على أنه يتم تبديل جزء من العمليات (التي تعتبر أقل استخداماً في الذاكرة الرئيسية) بين الذاكرة الرئيسية والقرص الصلب بهدف تحسين أداء الذاكرة الرئيسية وذلك بتقليل العمليات الموجودة في الذاكرة الرئيسية .

إذن البادلة هي عبارة عن حركة العمليات بين القرص والذاكرة الرئيسية ؛ و هذه الحركة تظهر عندما يكون هناك مقدار من الذاكرة الحقيقية ليست ذات أهمية عالية و عمليات ليست نشيطة فتنتقل من الذاكرة الرئيسية إلى فراغ المبادلة لتوسيع الذاكرة الرئيسية0

في هذا الموضوع سوف نتحدث عن ذلك الجزء المقتطع من القرص الصلب لأجل عملية التبديل والذي يُدعى : (Swap-Space)

(swap-space)فضاء التبديل هو مساحة مستقطعة من القرص الصلب - ذاكرة تخيلية لتوسيع الذاكرة الرئيسية - مخصصة لعملية التبديل ، أي يمكن اعتباره جزء مكمل للذاكرة الرئيسية .

الswap space)بما أن الذاكرة الافتراضية تقوم باستهلاك مساحة من القرص الصلب ، فإن عملية الوصول الى القرص (ستقلل من أداء النظام .

لنظام الذاكرة التخيليةthroughput) منتوج (   ) هو توفير أفضل swap space عملية التبديل (اذا، فالهدف الرئيسي من و ليست سرعة عملية الوصول الى الذاكرة. , يُستخدم بطرق عدة اعتماداً على إدارة الذاكرة المستخدمة في نظام التشغيل (swap space) فضاء المبادلة فعلى سبيل المثال:

يخزن الصفحات التي تم إخراجها من الذاكرة الرئيسية.(Paging system)نظام الملفات كمية المساحة المستخدمة في التبديل في أي نظام تعتمد على حجم الذاكرة الفعلية في النظام، بالإضافة الى حجم الذاكرة التخيلية التي يدعمها هذا النظام , وطريقة إستخدام هذه الذاكرة التخيلية.

مسبقا.مثلاً: (sawp file) لذلك من المهم تحديد حجم ملف المبادلة

. RAM نظام لينكس ينصح بأن يكون حجم هذا الملف ضعف الذاكرة

:(2-6-12 Swap-Space Location **مواقع فضاء المبادلة**)

لها موقعان :

- إما يكون داخل نظام الملفات (file)

وفي هذه الحالة تكون مقتطعة من نظام الملفات و يتم التعامل معه معاملة الملفات.

من مساوئها ان عملية البحث في دليل الملفات يأخذ وقت وعدد كبير من عمليات الوصول الى القرص,

يزيد من عدد مرات التبديل. (external fragmentation) فان التقسيم الخارجي بالاضافة الى ذلك

➢ أن يكون منفصل (disk partition)

في هذه الحالة تكون قسم مستقل من القرص الصلب لا تقل مساحته عن 20% من مساحة القرص الصلب.

في هذه الحالة يجب أن تتم عملية Mount

و هو أفضل وأكثر شيوعا0

(مثال على إدارة فضاء المبادلة 3-6-12 Swap-Space Management :Ex:)

أنواع فضاء المبادلة (swap space):

هناك ثلاثة أنواع من فضاء المبادلة ،وكل نوع مستعمل بشكل مختلف بالنظام وله حسناته وسيئاته..

➢ فضاء تبادل الجهاز(Device swap) :

فضاء تبادل الجهاز (Device swap space) يحتل حجم أو تقسيم منطقي، حيث أنه يُحجز بشكل واضح لتبديل الأغراض.

وهذا الفضاء (space) قد يّشكل على أنه منطقة نفايات (dump area).

تبادل الجهاز (Device swap) يمكن أن يُستخدم محلياً فقط . ولا يمكن للعميل (clien t) الذي يستخدم NFS أن يدخل إليه عن بُعد .

تبادل الجهاز (Device swap) يمكن الوصول إليه بسرعة لأن نظام التشغيل يمكن أن يصل إلى الحجم المنطقي أو يقسم بسرعة لأداء أكبر عمليات إدخال وإخراج.

- تبادل نظام الملفات (File system swap) :

تسمح بالتبادل الإضافي إذا كان هناك حاجة أكثر من تخصيص فضاء تبادل الجهاز (Device swap space). ويستعمل تبادل نظام الملفات (file system swap) فقط إذا كان (device swap space) غير كافي..

عندما يحتاج النظام إلى تبادل إضافي فإن تبادل نظام الملفات (file system swap) يسمح باستعمال فراغ نظام الملف الحالي بدلاً من حجز كامل الحجم أو التقسيم المنطقي ؛ لأن تبادل نظام الملفات يتطلب من النظام أداء كمية أكبر من المعالجة وهي في العادة أبطأ من فضاء تبادل الجهاز (device swap) . ولذلك يجب أن لا يُستخدم كبديل دائم لفضاء تبادل الجهاز device swap space) .

تبادل نظام الملفات يستخدم لتبادل داخلي (محليّ) أو بعيد ، حيث أن مجموعة من العملاء (cluster clients) يمكنهم أن يستخدموا تبادل نظام الملفات البعيد لتبادل حاجاتهم.
التبديل إلى نظام الملفات البعيد أبطأ من التبديل إلى نظام الملف المحليّ ,
ولا يُنصح به إذا كان تبادل الجهاز(device swap) المحليّ أو نظام الملف المحلي متوفر.

- فضاء التبادل المُزيّف (Pseudo-Swap) :

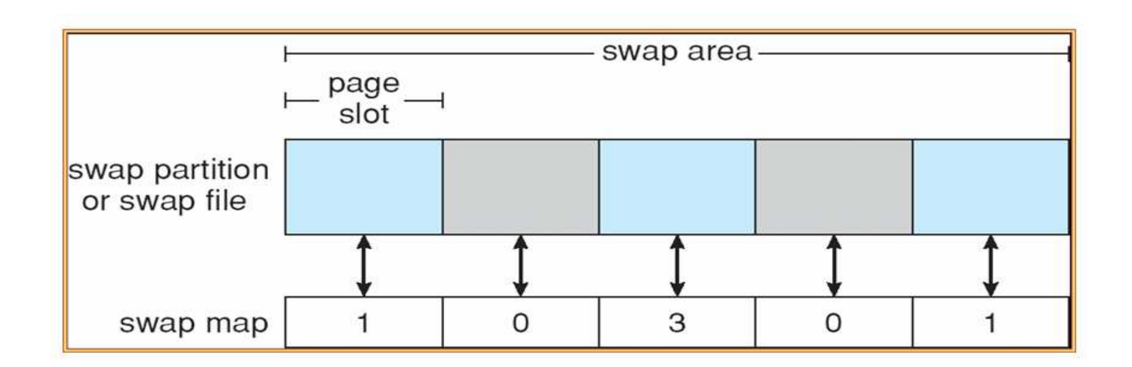
عند استعمال فضاء التبادل المُزيّف (pseudo swap) ، يمكن خلق عمليات أكثر، ويمكن كذلك زيادة تحميل النظام ، وذلك يسبب تبديل الصفحات (paging) أكثر ويعطل النشاط.

و يكون فضاء التبادل المُزيّف (pseudo swap space) متوفر ، لكن إذا لم نكن نريد استعماله فإننا نحتاج إلى إعادة tunable system parameter ، swapmem_on, to 0 (off)

(مثال على إدارة فضاء المبادلة 4-6-12 Swap-Space Management :Ex:)

نظام اللينيكس مشابه لنظام السولاريس بأنه مستخدم لأجزاء من الذاكرة مشتركة بين عمليات (معالجات) مختلفة.

. يكون ضعف الذاكرة الرئيسية swap-space) فضاء المبادلة (Linux في نظام الــ



أنظمة لينيكس   تركيب البيانات لأجل المبادلة على

نظام لينيكس يسمح بوجود مجموعة من فراغات المبادلة و كل فراغ مبادلة يتكون من 4 كيلو بايت تُدعى :

وهي تستخدم لإمساك الصفحات المستبدلة .(page slot).

واحدة أو أكثر.) (swap areaو هو يسمح كذلك بأن يكون هناك منطقة تبديل

. (Page slots)تحتوي على سلسلة من (swap area) حيث أن كل منطقة تبديل

و لكل منطقة تبديل خريطة مبادلة خاصة بها و هي عبارة عن مصفوفة من العدادات :

- متواجدة .(page slot) أن إذا كانت قيمة الرقم (العداد) تساوي صفر فهذا يدل على
- أما إذا كان الرقم (العداد) أكبر من الصفر
- (page slot) فإذا كانت قيمة الرقم (العداد) تساوي 1 فهذا يدل على أن

مشغولة من قِبَل صفحة مبادلة .

- أما إذا كانت قيمة الرقم (العداد) تساوي 3 فهذا يدل على :

أن هناك 3 عمليات مختلفة تؤشر على هذه الصفحة المستبدلة (swapped page).

(حيث يكون في هذه الحالة الرقم (العداد) ذاته يدل على عدد العمليات التي تستخدم هذه الصفحة ؛ فالقيمة 4 مثلاً تدل على أن هذه الصفحة يؤشر عليها 4 عمليات).

مشاركة الطالبات:

منيرة عبدالله بن هريس – ريهام حافظ– رامه البلوي – نورة الزيد – هنادي الحزيمي

المراجع:

- **operating system concepts**
- **http://docs.hp.com/en/B2355-90672/ch06s02.html**

الفصل العاشر:

أنظمة الإدخال والإخراج

أنظمة الإدخال والإخراج
(I/O system)

نظام المدخلات والمخرجات

يقصد بعمليات الإدخال والإخراج تبادل المعلومات بين وحدات الإدخال والإخراج ومسجلات وحدة المعالجة المركزية أو بين وحدات الإدخال والإخراج والذاكرة الرئيسية.

ومن الطرق التي تنفذ عمليات الإدخال والإخراج هي:

1. استخدام طريقة المسح بحيث لكل وحدة إدخال أو إخراج مفتاح معين يميزه (صفر أو واحد) يحدد فيها المبرمج منافذ الإدخال والإخراج المراد استخدامها وذلك من خلال تعليمات مباشرة.
2. الإدخال والإخراج باستخدام مفهوم الاعتراض interrupt (الاعتراض هو حدث استثنائي يضطر نظام التشغيل عند الاستجابة له وقف تنفيذ العمل الخاضع للتنفيذ من أجل تنفيذ عمل جديد يسمى العمل المعترض , وتمتلك نظم التشغيل برمجيات خاصة لمعالجة الاعتراض تسمى بمعالج الإعتراض (interrupt handler)).

وفيما يلي عناوين المواضيع التي سنتكلم عنها بشئ من الإيجاز لتوضيحها بإذن الله...

1. إن الكمبيوتر يعتمد على المدخلات والمخرجات مع الإجراءات التي تتم بداخله.
2. أجهزة الإدخال والإخراج.
3. تطبيقات المدخلات والمخرجات.
4. تحويل متطلبات المدخلات والمخرجات إلى عمليات على الأجهزة.
5. الأداء.

أولا: إن الكمبيوتر يعتمد على المدخلات والمخرجات مع الإجراءات التي تتم بداخله:

أ.الوظائف المهمة مثل: تصفح الويب أو تحرير ملــــــــــــــف.

ب.أن نظام التشغيل يدير أجهزة الإدخال والإخراج والعمليات المختلفة.

ج . أن هناك طائفة واسعة من آلات المدخلات والمخرجات.

د. البرامج والأجهزة التي تكون الكمبيوتر.

ثانيا:أجهزة الإدخال والإخراج:

هناك أشكال متعددة وأنواع مختلفة للأجهزة المستخدمة في الإدخال والإخراج:

- (Port - a connection point) المنافذ ونقاط الاتصال.
- (Controller)الأجهزة المتحكمة والمحولات.
- الأجهزة الناقلة وهي نوعين:

أ. (Memory-mapped I/O)الذاكرة التي تستخدم لتخزين الأشياء المراد نقلها.

ب.(Direct I/O instructions) نقل مباشر.

- الوصلات والأسلاك التي تربط بين الأجهزة المختلفة.

ثالثا:تطبيقات المدخلات والمخرجات:

مثل تطبيق الانتقال من الأجهزة لنظام التشغيل.

تختلف الأجهزة في كثير من الأبعاد و التي تؤثر في التطبيق:

- (Character-stream or block)طابع التيار أو الكتلة
- (Sequential or random-access) التسلسل والترتيب أو الوصول العشوائي.
- (Synchronous or asynchronous)التزامن في التوقيت أو عدم التزامن.
- (Sharable or dedicated) التقاسم والتشارك أو التخصص في شئ والتكريس.
- (Speed of operation) السرعة في العمليات والتشغيل.
- أنه يكون للقراءة والكتابة أو قراءة فقط أو كتابة فقط. (read-write, read only, or write only)

رابعا:تحويل متطلبات المدخلات والمخرجات إلى عمليات على الأجهزة:

نظره بشكل عام في قراءة ملف من القرص التي تتم عليه العملية....

1. تحديد الجهاز الذي سيحمل الملف.
2. ترجمة الاسم للجهاز الممثل .
3. قراءة البيانات من القرص العازل (buffer) إلى (disk)فيزيائيا.
4. جعل البيانات متاحة لمتطلبات العملية.
5. إعادة المراقبة للعملية.

خامسا:الأداء:

لتحسين الأداء:

1. خفض عدد المفاتيح(switches).

2. الحد من نسخ البيانات.

3. خفض الانقطاع.(interrupts) باستخدام تحويلات كبيرة.

4. توازن وحدة المعالجة المركزية (CPU),الذاكرة,الناقل.

كتبته:

ناديا العتيبي

المصدر

- Operating System Concepts by Silberschatz, Galvin and Gagne

الإدخال والإخراج في نظام اللينكس

يوجد طريقين في نظام اللينكس لمعالجة المدخلات والمخرجات داخل النظام

بالطريقة الأولى نستخدم نظام المناداة:

مثل: open(), read(), write()الخ

بينما بالطريقة الأخرى:

تستخدم مكتبة مناداة ال ANSI C

مثل: fopen(), fread(), fwrite() الخ

إن مكتبة السي في الحقيقة تحيط حول نظام المناداة و إنها تدعم قدر من الفوائد :

أولا: إنها تخفف الإخراج اتوماتيكيا , لذلك تقلل احتياجات المناداة لنظام المناداة الذي يحسن الأداء.
ثانيا: الوظائف الملائمة متوفرة مع مساعده بطريقة تعديل المخرجات قبل عرضها .

قسمت المخرجات والمدخلات إلى نوعين :

1.(System call I/O) نظام الإدخال والإخراج.

2.(stream I/O) طريقة جيدة جدا فهي تحسن أداء التطبيقات .

شروط الأخطاء:

على المرء إن يركز على اكتشاف الأخطاء ومعالجتها أثناء التعامل مع المدخلات والمخرجات فبرنامجك قد يعمل (بدون أي أخطاء) لكن عندما يحدث خطأ ما فإنه سوف يفسد برنامجك و لمنع حدوث الخطأ وخسارة البيانات من أخطاء المدخلات والمخرجات , وهي شروط الأخطاء هناك طرق كثيرة منها:

1Open(). تحتاج إلى فحص القيمة المرجعة من مناداة الوظيفة للتأكد من أن الملف المراد تم فتحه بالفعل.

2. write().فحص القيمة المرجعة من الوظيفة لنتأكد أن القرص الذي يُكتب فيه , لم يمتلئ أثناء كتابتك للبيانات.

ولكن عملية الفحص بعد كل نظام مناداة عملية مملة جدا...و لجعل عملية فحص الأخطاء ببرنامجك أوتوماتيكية تستطيع الذهاب إلى وظائف تفحص برنامجك وتكتب رسالة تخبرك بحدوث الخطأ إذا اكتشفه.

عدم عرقلة المدخلات والمخرجات:

دائما في نظام المناداة عندما ننادي وظيفة معينه , تنتظر إلى أن يتنفذ الآكشن المطلوب عمله هذه الطريقة في نظام الإدخال والإخراج تسمى بعدم عرقلة المدخلات والمخرجات ومن المهم أن نتذكر أن في هذه الطريقةِ، يَعُودُ نداءَ الوظيفةَ فوراً بصرف النّظر عن إكمالِ العملِ المطلوبِ.

الذاكرة – خرائط المدخلات والمخرجات:

يعتبر معلما هاما من معالم لينكس فمع هذه الميزة سيكون الملف مرتب بشكل حرفي في منطقه بالذاكرة ، فعندما تدخل تلك الذاكرة ، فإن العملية المخصصة تؤدي تلقائيا إلى الملفِ المحدد وهكذا. فباستخدام هذه الخرائط تزيد السرعة بشكل ملحوظ. لذا ، فإن هذا الأسلوب من المدخلات والمخرجات هو المفضل لدينا عندما تقرأ بيانات بالجملة.

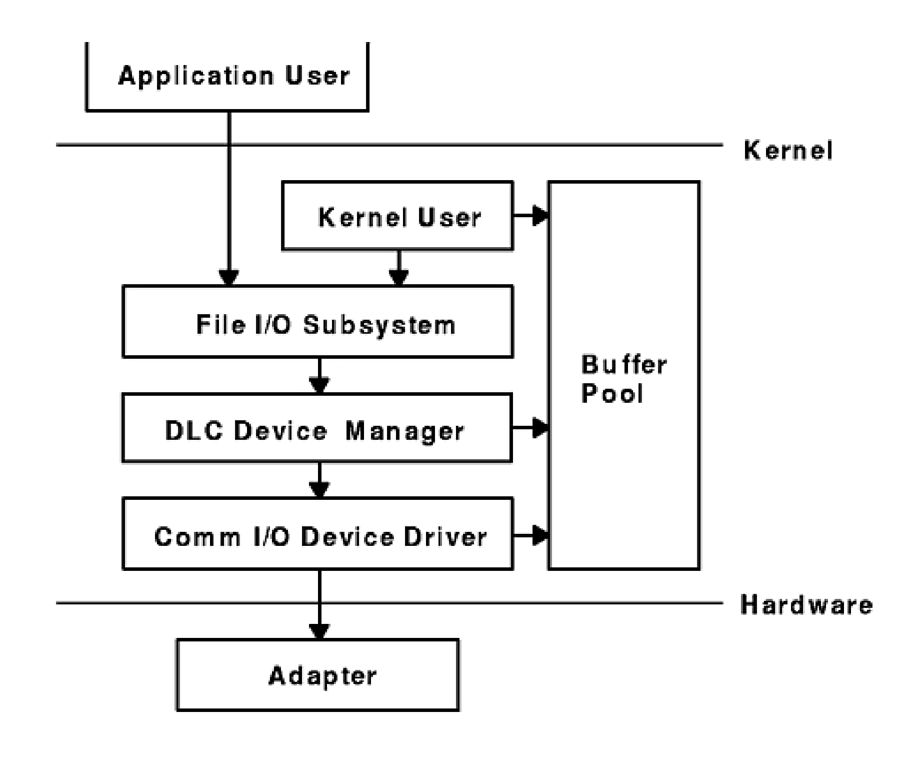
كتبته:

شيخة الرشود

المصدر:

http://www.enterprisenetworksandservers.com/monthly/art.php?1503:

إدارة وحدات الإدخال والإخراج



DLC Device Manager Environment

وحدات الإدخال والإخراج تتضمن الوحدات الحقيقية مثل آلة الطباعة,الأشرطة المغناطيسية والأقراص المغناطيسية, وكذلك الوحدات مثل وحدة التحكم, القنوات, وحدات تحويل.

وتشرف على هذه الوحدات برمجيات خاصة تسمى برمجيات إدارة وحدات الإخراج والإدخال (Device management).

حيث تأخذ هذه البرمجيات على عاتقها تنفيذ المهام والوظائف التالية :

1. متابعة وحدات الإدخال والإخراج وذلك من خلال استخدام كتلة وحدة التحكم لكل وحدة

(control block unit(CBU)) التي تتضمن كافة المعلومات المطلوبة عن كل وحدة إدخال وإخراج متصلة بالكمبيوتر.

2. اتخاذ سياسة معينة لحجز وحدات الإدخال أو الإخراج وفي زمن معين ولفترة زمنيه محددة وغالبا ما تعتمد هذه السياسة على طريقة استخدام وحدة الإدخال والإخراج(كوحدات غير مشتركة,مشتركة,إضافية)
3. الحجز الفيزيائي لوحدة الإدخال أو الإخراج وربط هذه الوحدة أو تلك مع العمل القابل للتنفيذ .
4. إعادة حجز وحدات الإدخال والإخراج المرتبطة مع العمل, وذلك بعد انتهاء تنفيذ العمل أو بعد انتهاء الفترة الزمنية المخصصة للحجز.

طرق استخدام وحدات الإدخال والإخراج:

تقسم وحدات الإدخال والإخراج حسب طريقة الاستعمال إلى:

1. الوحدات غير المشتركة (Dedicated I/O Devices)
2. الوحدات المشتركة (shared I/O Devices)
3.الوحدات الافتراضية (Virtual I/O Devices)

الواحدت غير المشتركة:

وهي وحدات يمكن حجزها لعمل واحد فقط. ولا يمكن اشتراك أكثر من عمل قابل للتنفيذ في استخدام هذه الوحدات, من الأمثلة على هذه الوحدات:

الآلات الطابعة,قارئ البطاقات المثقبة , وعند استخدام أي من هذه الوحدات من قبل عمل معين فان الأعمال الأخرى القابلة للتنفيذ والتي قد تحتاجها الوحدة المجوزة عليها الانتظار حتى تتحرر الوحدة المطلوبة عند انتهاء تنفيذ العمل أو عند انتهاء الفترة الزمنية المخصصة للعمل.

الوحدات المشتركة:

وهي وحدات يمكن استخدامها من قبل أكثر من عمل قابل للتنفيذ, وفي نفس الفترة الزمنية, ومن الأمثلة على هذه الوحدات : وحدات الإدخال والإخراج ذات الوصول المباشر
(Direct Access Storage Device(DASD)).

وتعتبر إدارة . وتعتبر إدارة مثل هذه الوحدات أصعب من إدارة الوحدات غير المشتركة نظرا للحاجة إلى تحديد أولوية عملية القراءة أو الكتابة للأعمال المختلفة القابلة للتنفيذ كالقرص المغناطيسي مثلا.

الوحدات الافتراضية:

وهي وحدات غير مشتركة تم تحويلها إلى وحدات مشتركة, وذلك باستخدام تقنيات وبرمجيات خاصة كنظام التمرير(spooling system) حيث يؤدي هذا النظام إلى تحويل الوحدة غير المشتركة إلى وحدة مشتركة, فمثلا يمكن استخدام القرص المغناطيسي كآلة طابعة, مما يؤدي بدوره إلى تحقيق الفوائد التالية:

1. حل مشكلة عدد الوحدات غير المشتركة فمثلا لو كان لدينا عدة أعمال تحت التنفيذ وكلها تحتاج إلى عمليات طباعة, فإنه لتنفيذ هذه الأعمال لابد من توفر عدد من الطابعات مساويا لعدد الأعمال المنفذة مما يؤدي بدوره إلى تكاليف باهظة, وباستخدام نظام التمرير يمكن الاستعانة بالقرص المغناطيسي حيث تتم عملية الطباعة على القرص, وعند الحاجة تنقل المعلومات من القرص المغناطيسي إلى الطابعة.
2. زيادة سرعة المعالجة نظرا لأن سرعة تعامل وحدة المعالجة المركزية مع القرص المغناطيسي أعلى منها عند تعامل الطابعة مع وحدة المعالجة المركزية.

كتبته:

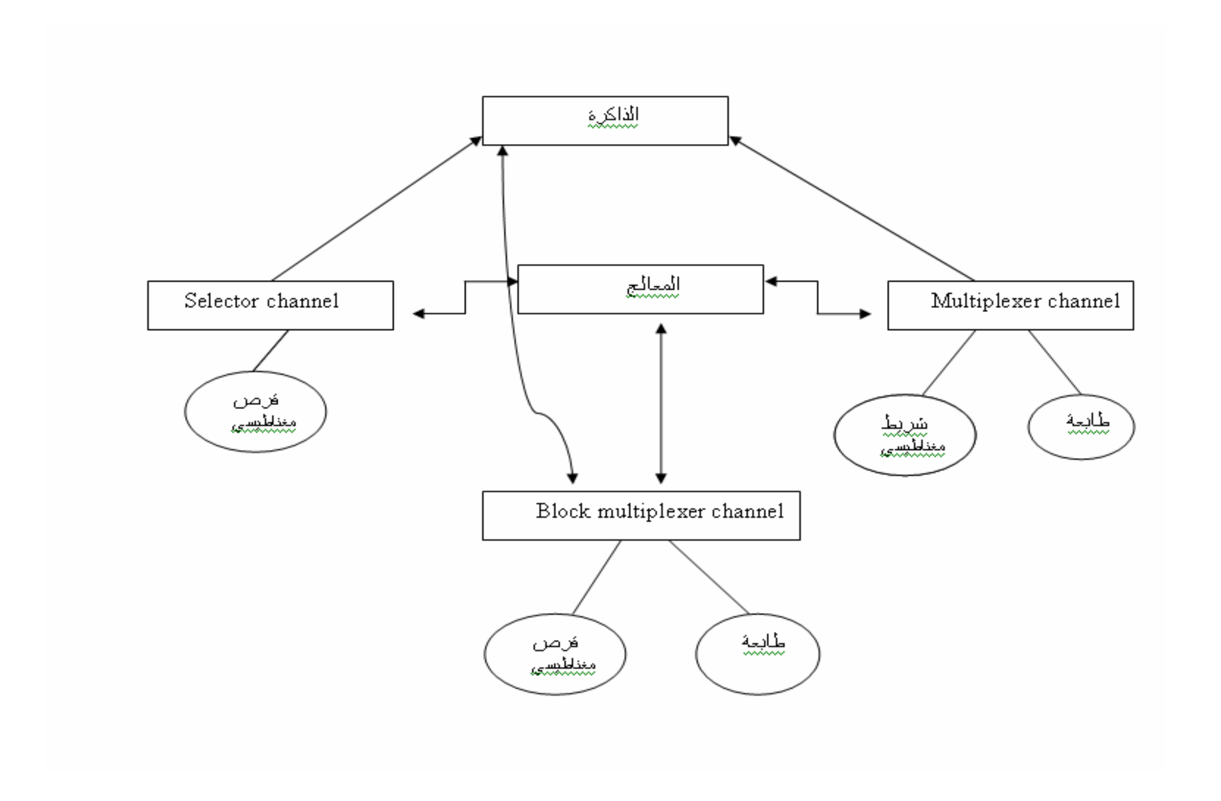
رزان المزروع

المصدر:

كتاب أنظمة التشغيل – للدكتور زياد القاضي.

قنوات الإدخال و الإخراج (IO\channels)

قنوات الإدخال و الإخراج عبارة عن أجهزة متخصصة يمكن أن تكون حاسوبا ، لكنها تتخصص في الإشراف على ربط وحدات الإدخال و الإخراج مع المعالج أو مع الذاكرة لإتمام عمليات الإدخال والإخراج .
وتحتوي هذه الأجهزة (القنوات) على برنامج خاص يسمى برنامج القناة (program channels)
ووظيفته متابعة عمليات الإدخال و الإخراج والاتصال بين وحدة الإدخال / الإخراج والمعالج أو الذاكرة.
وتوضح الرسمة كيفية استخدام قنوات الإدخال و الإخراج لإجراء عمليات الربط اللازمة .



المصدر:

كتاب أنظمة التشغيل للدكتور زياد القاضي

المقاطعة (interrupt)

تعريفه : هو عبارة عند حدث في الوحدات الصلبة للجهاز, المساعدة في عمليات معالجة البيانات محفز للانتقال من مكان في البرنامج الحالي في العملية إلى مكان معين آخر.

طريقة الinterrupt الأساسية:

تحتوي الوحدة المعالجة المركزية على سلك يدعى interrupt-request line بحيث تتحسسه بعد الانتهاء من كل أمر.فحين يكتشف المتحكم بالوحدة المعالجة المركزية أشارة على السلك,تخزن وحدة المعالجة المركزية الحالة الحالية وتنتقل إلى "مدير المقاطعة" interrupt handler وظيفة في عنوان محدد بالذاكرة .ويحدد interrupt handler سبب حدوث هذه المقاطعة(أي ظهور الإشارة) ,وتقوم بعمل العمليات الضرورية وتخزين الحالة والانتهاء من العملية والعودة إلى العملية السابقة التي كانت قبل ظهور الإشارة.وبالتالي فعليا يقال الجهاز المتحكم raises "تحفز" المقاطعة عن طريق إحداث إشارة في السلك.ووحدة المعالجة المركزية catches "تقبض" المقاطعة وبالتالي ترسل "dispatch" إلى interrupt handler وهو بدوره يزيل clear المقاطعة عن طريق أداء الخدمة للجهاز .

في أنظمة التشغيل الحديثة نحتاج إلى مواصفات إضافية للinterrupt handler منها :

1_ نحتاج إلى القدرة على تأجيل interrupt handler خلال العمليات المهمة .
2_ نحتاج إلى طريقة ذات كفاءة عالية في إرسال ا ل interrupt handler الملائم للجهاز من غير عمل polling "التصويت" لجميع الأجهزة لمعرفة أيا منهم قام بتحفيز المقاطعة.
3_ نحتاج إلى interrupt ذو عدة مراحل حيث نظام التشغيل يستطيع التمييز من له الأولوية الأعلى عن المنخفضة حتى يستجيب للمقاطعة بالدرجة المناسبة له.

*في الأجهزة الحديثة قاموا بتزويدها بثلاث المواصفات الذي ذكرت سابقا, المزودة أصلا عن طريق الوحدة المعالجة المركزية وعن طريق الوحدة الصلبة المساعدة في معالجة البيانات للمتحكم بالمقاطعة "توقيف".

حيث يوجد microprocessorsفي الأجهزة الحديثة تقوم بمعالجة عملية المقاطعة .
فحينما نحدث عملية مقاطعة في احد أجهزة الجهاز فإن وحدة المعالجة المركزية تتوقف عن التنفيذ ثم تقفز إلى موقع الذاكرة التي تحوي رمز المقاطعة أو أمر رمز المقاطعة .عادة ما يشتغل هذا الرمز في نمط خاص من وحدة المعالجة المركزية .
تصنف بعض CPU المقاطعات في درجات من الأولوية حسب حدوثها .
كذلك تخصص بعضCPU سجلات خاصة للمقاطعات A special set of registers.
وبعدما تتم عملية المعالجة للمقاطعات فإن يتم استعادة لحالة CPU السابقة بحيث يكمل CPU عملياته التي توقف عنها بسبب المقاطعة

كتبته
ماجدة بن طالب

المصدر:
http://www.science.unitn.it/~fiorella/guidelinux/tlk/node81.html
:

المقاطعة (interrupt)

الــ interrupt (مقاطعه):هو حدث في أجهزة ال hardware يجعل المعالج يقفز في برنامجه الحالي إلى نقطة محددة في هذا code.

*الــ interrupt (مقاطعه): صممت لتكون أحداث خاصة لا يمكن التنبؤ بها بدقة (أو على الإطلاق).
* MSP تحتوي على العديد من الأنواع المختلفة من الأحداث التي يمكن أن تؤدي إلى interrupts ، وكل واحد من المعالجات سيرسل تنفيذ فريد إلى نقطة محددة في الذاكرة

يمكن تقسيم الــ interrupt (مقاطعه) بصفة عامة إلى نوعين :

1- maskable

2 non-maskable

Maskable interrupt : هي المقاطعة التي تؤدى إلى توقف عجلة الأحداث ولكنه ليس مهم دائما ، لذلك المبرمج يمكن أن يقرر إن هذا الحدث يجب ألا يسبب قفز إلى البرنامج.

A non-maskable interrupt: مهم جدا و لا ينبغي أبدا أن نتجاهله, المعالج دائما سيقفز إلى هذاinterrupt(مقاطعه) عند حدوثه.

* في كثير من الأحيان ، maskable interrupt يتم إيقافه افتراضيا لتبسيط السلوك الافتراضي للجهاز.
*الــ interrupt (مقاطعه) عموما لها "الأولوية " عندما يحدث مقاطعتين (2 interrupts) في الوقت نفسه ، فإن الــ interrupt (مقاطعه) التي تملك أولوية أعلى سوف تكون لها الأسبقية على interrupt(مقاطعه) التي لها أولوية أدنى.

كتبته

غادة العندس

المصدر:

:http://cnx.org/content/m12321/latest/

أغلب وحدات المعالجة المركزية تحتوي على سلكين طلب لل interrupt :

1- الأول يدعىnonmaskable interrupt:

وهو محجوز للحدث مثل أخطاء الذاكرة الغير مصححة.

2- الثاني يدعى maskable interrupt:

وهو يمكنه الإيقاف عن طريق الوحدة المعالجة المركزية قبل الانتهاء من الأوامر المهمة التي من المفروض عدم إيقافها.وهو يستخدم عن طريق المتحكم بالجهاز لطلب الخدمات.

كتبته

زاهية الحربي

المصدر

Operating System Concepts by Silberschatz, Galvin and Gagne 7th edition

المقاطعة (**interrupt**)

Interrupt request line:

هو سلك بين CPU & controller يقوم CPU بفحصه بعد تنفيذ كل أمر لمعرفة ما إذا controllerقد وضع فيه شحنة ما

عندما يجد CPU إن هنالك شحنة على السلك فعندها يعمل ما يعرف ب context switchفيعمل حفظ لما كان يعمله ثم يذهب إلى **Interrupt handler** الموجود في مكان ثابت في memory

Interrupt handlerيقوم بتحديد حالة Interruptو يقوم بعدة عمليات ضرورية لمعالجتهم ثم يرجع إلى CPU ليكمل ال CPU ما كان يفعله وتستمر الدورة .

يوجد لدينا نوعان من أسلاك **interrupt**:

1- Non-maskable interrupt:

هذا النوع لأجل الأخطاء التي يجب معالجتها في الحال مثل عنوان في الذاكرة لا يسمح بالدخول له.

2.Maskable

يستطيع CPU أن يقفل هذا الخط لكي لا يتم إزعاجه خلال قيامه بتنفيذ الأوامر وهو يستخدم عادة لخدمة أجهزة الإخراج والإدخال.

Interrupt vector:

يحتوي على عناوين في الذاكرة لمجموعة مميزة من Interrupt handler وذلك لتقليل من عملية البحث ما إذا كان هنالك يوجد فقط handlerواحد يقوم بالبحث عن جميع مصادر interruptلمعرفة من يريد الخدمة.

Interrupt priority levels:

هذه التقنية تمكن CPU من التفريق بين المقاطعات ذات الأهمية العليا والتي ذات الأهمية الدنيا

نظام التشغيل يتعامل مع تقنية المقاطعات بعدة طرق:

1. Boot time فان CPU يحتاج لمعرفة من المتصل به من أجهزة الإخراج والإدخال وذلك لتعبئة Interrupt vector
2. Exception وهي الأخطاء التي تأتي من Software مثل القسمة على صفر .

كتبته

نوف الغانمي.

المصدر:

: Operating System Concept

المقاطعة (**interrupt**)

مفهوم الاعتراض واستخدامه في عمليات الإدخال والإخراج:

الاعتراض هو حدث استثنائي يضطر نظام لتشغيل عند الاستجابة له وقف تنفيذ العمل الخاضع للتنفيذ من أجل تنفيذ عمل جديد يسمى العمل المعترض

وتمتلك نظم التشغيل برمجيات خاصة لمعالجة الاعتراض تسمى بمعالج الاعتراض interrupt handler وتتولى هذه البرمجيات تنفيذ الوظائف التالية:

1..إصدار الاستجابة الموجبة والسالبة بناء على طلب الاعتراض من وحدة الإدخال والإخراج.

2.وفي حالة تقبل طلب الاعتراض عليه حفظ حالة البرنامج الخاضع للتنفيذ ولنسمه العمل القديم وذلك للرجوع إليه لاحقاً

3.يتم تحميل حالة العمل المعترض الجديد بعد وقف تنفيذ العمل القديم.

4.ينفذ العمل الجديد

5.بعد الانتهاء من العمل الجديد تحمل حالة البرنامج القديم لمتابعة تنفيذه من النقطة التي قطع منها.

اعتراض الإدخال والإخراج I\O interrupt:

وينشأ هذا الاعتراض في قناة الإدخال أو الإخراج أو وحدات الإدخال والإخراج وأسبابه واحد مما يلي أو أكثر:

1. خطأ في أمر الإدخال أو الإخراج كأن يكون الأمر غير صحيح لغوياً أو غير متبع لقواعد كتابة أوامر الإدخال أو الإخراج أو خطأ في شفرته الثنائية أو عدم تحديد هذا الأمر كلا من رقمي وحدة الإدخال والإخراج ورقم القناة.

2. انتهاء تنفيذ برنامج الإدخال أو الإخراج وذلك بعد تنفيذ كلمة command word channel في برنامج القناة حيث يصدر نظم التشغيل الأمر المهيمن HIO لإيقاف برنامج القناة وبذلك يحدث اعتراض مفاجئ على القناة لتتوقف وتعمل أخرى في الحاسوب.
3. انتهاء وحدتي الإدخال أو الإخراج من عملية تمرير ونقل البيانات وفي هذه الحالة يتحكم برنامج ضابط التحكم بالإدخال والإخراج من ضبط عملية نقل البيانات بالاتجاهات المختلفة وذلك بالتعاون مع نظام التمرير Spooling system فإذا تمت عملية تمرير البيانات حدث اعتراض مفاجئ.

المصدر:

أنظمة التشغيل للدكتور زياد القاضي

الوصول المباشر للذاكرة (Direct Access Memory)

Direct Memory Access: هو نظام يسمح بنقل البيانات مباشرة من الذاكرة إلى أنظمة الإدخال والإخراج دون الحاجة لإذن من ال CPU.

Module: تنقل البيانات من موقع ذاكرة إلى موقع ذاكرة آخر.

* حالات الوصول إلى الذاكرة بشكل آلي أسرع بكثير من أن تدير CPU الانتقالات.

* الأنظمة ADC, DAC, PWM, تأسر كل متطلبات تحركات ال memory المتكررة والمنتظمة خارج أنظمتهم الخاصة بهم ..

* DMA : يمكن استخدامها لمعالجة انتقال البيانات المجمعة خارج الوحدات الخارجية إلى مواقع ذاكرة مفيدة أكثر.

Memory هي الوحيدة التي يمكنها أن تدخل هذا الطريق، لكن أكثر الأنظمة الخارجية peripheral systems وسجلات البيانات data registers والسجلات المتحكمة control registers يمكنها أن تدخل هذا الطريق كأنه ذاكرة..
DMA تستخدم عندما تكون الكهرباء منخفضة لأنها تستخدم نفس memory bus كوحدة المعالجة المركزية وفقط واحد أو الآخرون يمكنهم استخدام memory في نفس الوقت.

DMA: منظم إلى ثلاثة أجزاء مستقلة.. ومع ذلك فإن الثلاثة تتنافس على نفس memory bus ويمكن أن يشكلوا ال independent triggers , memory regions

هناك ثلاث قنوات مستقلة لانتقالات DMA .. كل قناة تستقبل trigger حتى ترسلها لأكبر عدد من ال signals المختارة.. عندما تكون هذه ال signals نشطة يتم الإرسال ..

DMA controller : يستقبل trigger signal ولكنه يهملها تحت شروط معينة.. وهذا ضروري لحجز memory bus لإعادة البرمجة و non-maskable interrupts
Controller يعالج التعارضات للsimultaneous triggers

كتبته

رهام حافظ

المصدر:

:/http://cnx.org/content/m11867/latest

.

مشكلة ال(DMA)مع الذاكرة الوسيطة:

إذا كانت وحدة المعالجة المركزية مجهزة بذاكرة وسيطة وذاكرة خارجية التي يمكن أن تُدخل مباشره بواسطة الأجهزة التي تستخدم (DMA)ـ.
عندما تدخل وحدة المعالجة المركزية الموقع (س) في الذاكرة فإن القيمة الحالية لـــ(س) سَتَخْزنُ في الذاكرة الوسيطةِ.
والعمليات اللاحقة على(س) ستُحدِث النسخة الموجودة في الذاكرة الوسيطة بينما النسخة الموجودة في الذاكرة الخارجية لم تُحدَّث بعد.
لو حدث وقرأ أحد الأجهزة الموقع (س) فقبل أن تحدث نسخة الذاكرة الخارجية فإنه سيقرأ فإن القيمة الغير محدثة.

وبنفس الطريقة لو كتب الجهاز في الذاكرة الخارجية ولم تُحدَث نسخة الذاكرة الوسيطة فإن وحدة المعالجة ستقرأ قيمة غير مُحدَّثة



كتبته

صفاء البسام

المصدر: http://en.wikipedia.org/wiki/Direct_memory_access

ذاكرة الوصول المباشر (DMA):

يقدم CPU الكثير من الأعمال , فهو يشغل Bios ونظام التشغيل والتطبيقات . كما يعالج المقاطعات وعناوين I/O وأيضاً يعالج CPU كل البيانات بشكل دائم إذ يقوم بنقلها من RAM إلى مكان آخر كما ترسل الطرفيات , مثل الطابعة والماسح الضوئي البيانات إلى RAM عبر CPU , وترسل CPU البيانات من RAM إلى الطرفيات .

ومن الواضح أهمية نقل هذه البيانات , لكنها عملية بسيطة في النهاية , ولدى CPU أعمال أكثر أهمية لتقوم به.

علاوة على ذلك , ومع كل هذه الذاكرة المخبئة Cache memory في CPU الحديثة يهدر النظام معظم وقته في انتظار معالجة بعض الحسابات الداخلية في CPU.

ولذلك يطرح السؤال التالي : لماذا لا يوجد جهاز يقوم بالوصول المباشر إلى الذاكرة دون تدخل CPU?

تدعى عملية الوصول المباشر إلى الذاكرة بدون تدخل CPU : Direct Memory Access

وهي سمة من سمات أجهزة الكمبيوتر الحديثة التي تسمح لبعض الأنظمة الفرعية داخل أجهزة الكمبيوتر للوصول إلى نظام لذاكره القراءة و / أو الكتابة بصورة مستقلة من وحدة المعالجة المركزية ((CPU))..

تتيح DMA تشغيل التطبيقات في الخلفية , بدون تدخل CPU . وهذا طبعاً ممتاز لتشغيل الصوت في الخلفية , ونقل البيانات من القرص المرن أو القرص الصلب إلى RAM.

العديد من أنظمة hardware تستخدم **DMA** والتي تشمل متحكمي تشغيل القرص ((disk drive controllers)) وبطاقات الرسوم وبطاقات الشبكة وبطاقات الصوت ..

أجهزة الكمبيوتر التي لديها **DMA** channels تستطيع نقل البيانات من والى الأجهزة بتكلفة أقل بكثير من وحدة المعالجة المركزية ((CPU)) ..

مع **DMA** , وحدة المعالجة المركزية ((CPU)) ستبدأ بالنقل , وإجراء العمليات الأخرى بينما النقل مستمر , وكذلك استقبال القاطع ((interrupt)) من متحكم الــ **DMA** عندما تكتمل العملية ,

هذا جدا مفيد وخصوصا في التطبيقات الحسابية للوقت الحقيقي ((real-time)) وبذلك لا يكون هناك تعطيل للعمليات المتزامنة ..

المشكلة هنا : ما العمل إذا طلب أكثر من جهاز استخدام DMA ؟ كيف نمنع تزاحم الأجهزة على ممر البيانات الخارجي ؟ ماذا لو احتاجت CPU إلى ممر البيانات فجأة ؟ كيف يمكن إيقاف جهاز يستخدم DMA بحيث تستطيع CPU(ذات الأولوية) الوصول إلى الممر ؟

ولمعالجة ذلك قامت IBM بإنشاء رقاقة تدعى رقاقة 8237 للتحكم بعمليات DMA.تستطيع هذه الرقاقة معالجة كل عمليات نقل البيانات من الطرفيات إلى RAM , أو بالعكس . وهذا ما يوفر من وقت وجهد الــــــ CPU .

كتبته

مي الغيث – هند المطيري

المصدر

http://en.wikipedia.org/wiki/Direct_memory_access:

الاستطلاع(polling)

ماهية فكرة الاستطلاع (Polling)؟

لها الكثير من المصطلحات ولكن نستطيع تشبيهها بالايميل . فعندما يصل الايميل للشخص فانه يتجمع في الصندوق الوارد . ويقوم بعدها الشخص (صاحب الايميل) بفحص أيميله كل فترة لمعرفة الجديد في صندوقه الوارد .

هذه فكرة مبسطه عن الاستطلاع (poll) فجهاز الكمبيوتر يحتوي على العديد من أجهزه الإدخال والإخراج المتصلة به وكلها تحتاج في وقت ما إلى محادثة المعالج .

عن طريق هذه الفكرة فان المعالج يقوم كل فترة زمنيه بفحص جميع أجهزة الإخراج والإدخال المتصلة به لمعرفة ما إذا هنالك جهاز يحتاج إلي خدمة ما فيقوم بخدمته

Polling

هو بروتوكول للتفاعل بين المضيف(CPU) و المتحكم (controller).

Controller يصف حالته عن طريق Busy Bit الموجود في status register

controllerعندما يكون مشغول يعمل set busy bit(write 1 on it) وعندما يكون جاهز لاستقبال الأوامر فانه يعمل clear to busy bit (write 0 on it)

هذا بالنسبة لل controller كيف يعبر عن حالته أما بالنسبة إلى CPU فانه يصف حالته إلى controllerعن طريق command-ready bit in command register

التعامل بينهم يكون كالتالي :

المضيف يفحص باستمرار Busy bit حتى يجده ب 0 فيعلم أن controller جاهز لاستقبال الأوامر المضيف يعمل set to write bit in command register ثم يكتب byte في data-out registerيعمل المضيف set command-ready bit

عندما يلاحظ controller بان ready bit has been set يقوم بعمل set to busy bit

ثم يقو ال controller بقراءة البيانات الموجودة في command register ويجد أن write قد عمل له set(قيمته ب 1)

فيعلم بان عليه أن يرسل هذه البيانات إلى جهاز إخراج

ثم يقوم ال controller بعمل مسح (clear) لل command ready bit وأيضا يقوم بمسح error bitلكي يعلم المضيف بان عملية الكتابة تمت بنجاح وأيضا يقوم بمسح busy bit ليعلمه بان مهمته انتهت ثم تقوم هذه الدورة بالاستمرار.

كتبته

نوف الغانمي

المصدر

: Operating System Concept